# Corelight-update

Updated Sep 18, 2023

# CONTENTS

The **primary purpose** of the Corelight-update utility is to automate and simplify the workflow of collecting data from disparate sources of dynamic content for Corelight Sensors by integrating into your existing CI/CD process.

This data includes threat intel, Suricata rulesets, vulnerability data, Zeek packages and other Input Framework data. The data can come from pre-formatted local sources, pre-formatted remote sources, or third-part integrations.

There's no need for additional tools if you integrate Corelight-update with your CI/CD or change control process to manage Suricata rulesets, Intel files, Input files or Zeek package bundles.

In addition to collecting and formatting data sources, Corelight-update can optionally apply Corelight best practices to Suricata rulesets, extracting indicators from atomic Suricata rules and creating Zeek Intel files. The corresponding Suricata rules are then disabled, reducing the workload of the Suricata process.

Corelight-update natively supports the concept of hierarchical processing with a single global configuration and multiple policy configurations.

The output of each policy is a single Intel file, a single Suricata ruleset, a single package bundle, and multiple Input files ready to be consumed by a Corelight Sensor.

A **secondary function** of Corelight-update is to push content to Corelight Sensors. It supports ALL types of sensors, both Fleet-managed and stand-alone.

CHAPTER

ONE

# CORELIGHT-UPDATE RELEASE NOTES

## 1.1 v1.8.1

### 1.1.1 Bug fixes

- Fixed the path for push published input files.

## 1.2 v1.8.0

### 1.2.1 Enhancements

- Added support for Suricata policies in Fleet Manager v27.8.
- Added configuration option to specify Fleet Manager sensor policy and suricata policy names.
- Added a feature to remove “disable” individual intel indicators.
- Added support for remote Input files.
- Added support to use the same Input file from multiple sources at the same time.
- Increased support for pushing in parallel to Fleet Manager managed sensors.
- Added support to auto replace autoupdate policy config files when the database structure changes.

## 1.3 v1.7.3

### 1.3.1 Enhancements

- Added additional fields to the CrowdStrike Hosts integration.

## 1.4 v1.7.2

### 1.4.1 Enhancements

- Added a network timeout variable for waiting on a status from a sensor after a file upload.

### 1.4.2 Bug fixes

- Fixed a bug that caused an exit if the icannTLD integration had an error.

## 1.5 v1.7.1

### 1.5.1 Enhancements

- Added a bash_completion script.

### 1.5.2 Bug fixes

- Fixed a bug so checking the status of an uploaded file through Fleet Manager uses a bearer token.
- Adding a new user in RPM based OS's adds a '/sbin/nologin' shell

## 1.6 v1.7.0

### 1.6.1 Enhancements

- Added a new integration for Crowdstrike Spotlight CVE.
- Added a new integration for Crowdstrike Spotlight Hosts.

## 1.7 v1.6.3

### 1.7.1 Bug fixes

- Fixed missing network settings after upgrade issue.

## 1.8 v1.6.2

### 1.8.1 Enhancements

- Improved logging output to log when a download attempt is intercepted by an external proxy.
- Added support to configure network settings for sensor communications.
- **Added the following new CLI options:**
  - `show -network`
  - `update -network-setting [setting1=value1 setting2=value2 ... settingN=valueN]`
  - `update -network-settings [setting1=value1 setting2=value2 ... settingN=valueN]`

### 1.8.2 Bug fixes

- Fixed a bug that caused sensor traffic to use the same proxy configuration as download traffic.
- Improved error output when updating policy configurations.
- Fixed a bug that compared suricata config files before they are downloaded.

## 1.9 v1.6.1

### 1.9.1 Enhancements

- Added support to prevent policies from being created if the name begins with a -.

### 1.9.2 Bug fixes

- Fixed an issue that prevented downloading intel sources for a policy without any suricata sources.

## 1.10 v1.6.0

### 1.10.1 Enhancements

- Added a global option to auto-update policies each time the service runs.
- Added a global setting to push content to sensors in parallel (defaults to 10).

## 1.11 v1.5.0

### 1.11.1 Enhancements

- Added support for uploading Suricata configurations to all sensors.
- Added support for pushing signed package bundles to all sensors, except Software Sensor v1.x systems.
- Added support for the new CSRF requirement in the Fleet Manager v27.3 API.

## 1.12 v1.4.1

### 1.12.1 Bug fixes

- Fixed an issue where empty options were written to Suricata rules.
- Fixed an issue where an empty "If-Modified-Since" header is used during file downloads.

## 1.13 v1.4.0

### 1.13.1 Enhancements

- Added a new integration for Mandiant Threat Intelligence.
- If Fleet Manager details are configured, and a matching policy exists, the Fleet Manager policy will be updated even if no sensors are assigned to it.
- **Added the following new CLI options:**
    - `add -policy` and `add -policies` are interchangeable.
    - `remove -policy` and `remove -policies` are interchangeable.
    - `-file` and `-path` are interchangeable on all relevant CLI commands.
    - **Most of the Global configuration settings can be updated directly from the CLI:**
        - `update -global-setting [setting1=value1 setting2=value2 ... settingN=valueN]`
        - `update -global-settings [setting1=value1 setting2=value2 ... settingN=valueN]`
- Added "basic" auth support for sources.
- **Added support for pulling Global Suricata config files from remote sources.**
    - Includes support for no auth, basic auth, and token auth.
- **Added support for pulling Policy Suricata config files from remote sources.**
    - Includes support for no auth, basic auth, and token auth.
- Added the ability to append content to the Metadata and Other fields using modify.conf.
- Added the ability to identify rules with Metadata contains string.
- Added the option to include disabled Suricata rules in the ruleset file.

- Simplified the global configuration by removing the global integration table. Each integration is now enabled using its own settings.
- The `update -policy` command now uses a transaction. If any part of the update fails, the update is not applied.
- Removed the config templates (obsolete). The `import -policy <policy name> -file <path to config file>` can be used to the same config to different policies.
- Removed the policy backup functions (obsolete). The `show -policy <policy name> -file <path to save config file>` can be used to save a backup.

### 1.13.2 Bug fixes

- Fixed a issue where package bundles were not created with `other:read` permissions on all files, causing packages not to load on sensors.
- Pushing package bundles now updates a Fleet Policy instead of trying (and failing) to push through Fleet to the sensors.

## 1.14 v1.3.0

### 1.14.1 Enhancements

- Fleet managed sensors no longer have to be listed in the inventory section of the policy. The list will automatically be pulled from Fleet Manager.
- Added support for AlienVault OTX.
- Added configurable URL for ICANNTLD.
- The Integration table has been removed, each integration is now enabled within it's configuration.

## 1.15 v1.2.1

### 1.15.1 Enhancements

- Added a basic web menu to the root of the webservice.

### 1.15.2 Bug fixes

- Fixed a bug that would cause a policy to fail if no intel files were present.
- Added a redirect to the webservice if the trailing slash is missing for `\docs\` or `\files\`.

## 1.16 v1.2.0

### 1.16.1 Enhancements

- Added support for global cache and policy level Intel sources that can be downloaded in Zeek format, like ThreatQ.
- Added support for Token authenticated Suricata and intel sources like MISP.
- Updated the web service to use TLS version 1.2+ and removed outdated cipher suites.

### 1.16.2 Bug fixes

- Improved error handling with TenableSC.
- TenableSC was not reading the keys from the policy in the database.
- Moved the home directory for the corelight-update service account to `/var/corelight-update/`
- Removed the requirement for experimental features to be enabled to upload Suricata rules to Fleet.

## 1.17 v1.1.0

### 1.17.1 Enhancements

- Support for encrypted passwords for inventory items.
- Corelight-update now uses a umask of `0007` when creating files and directories.

### 1.17.2 Bug fixes

- The before-install and before-upgrade scripts will not attempt to create the system user if it already exists.
- Downloading content will now use the `https_proxy` or `HTTPS_PROXY` environment variables.

## 1.18 v1.0.1

### 1.18.1 Enhancements

- Policies are stored in a Sqlite3 DB".
- The Corelight-update service now runs as corelight-update and not root.
- After install or upgrade, all files are owned by system user corelight-update:corelight-update.
- All users must belong to the `corelight-update` user group to run Corelight-update.
- Global configuration can be updated from either a `yaml` or `json` config file.
- Policies configurations can be imported or updated from either a `yaml` or `json` config file.
- Sources that do not require authentication can be added as type "suricata" or "intel".
- A Global Source Cache is automatically created.

- Integration intervals are now referenced in `hours` See *Third-party integrations settings* for details.
- The interval for processing policies is now referenced in `minutes` See *Configuration settings* for details.
- The web Service no longer requires root privileges to enable ports below 1024.
- **Pushing Suricata rulesets to Fleet managed sensors no longer proxies that push through Fleet.** It uploads the ruleset to Fleet and updates the Fleet policy to use the new ruleset.
- **When pushing content to sensors, an inventory file is no longer used.** The sensor details are part of the policy config.
- Missing configuration files are automatically recreated.

### 1.18.2 Bug fixes

- Set http.Transport idelConnTimeout for Fleet to 90 seconds.

CHAPTER

TWO

# QUICKSTART - NEW INSTALL

The Corelight-update utility can run as a service at a scheduled intervals, if enabled, to check for updates to the configured data sources, and distribute updated content to the Fleet Manager policies and sensors.

## 2.1 System requirements

The minimum system requirements are:

- An x86_64 or ARM64 processor.
- 4 GB memory.
- A host running a Linux OS.
- Network connectivity to the Internet, or to an internal-facing threat intelligence data repository.
- To push content to your sensors, or to Fleet Manager, network connectivity to the management interface is required.

## 2.2 Installation overview

Select a host to install the Corelight-update utility. If you have a Corelight Fleet Manager installation, Corelight-update can be run on the same host.

*1. Set up the Corelight stable package repository*

*2. Install Corelight-update*

*3. Add the corelight-update group to existing users (optional)*

*4. Configure Corelight-update (optional)*

*5. Add proxy configuration (optional)*

*6. Run Corelight-update*

### 2.2.1 1. Set up the Corelight stable package repository

#### Bash script - deb Installation

1. Run the script using:

   ```
   curl -s https://packages.corelight.com/install/repositories/corelight/stable/
   ↪script.deb.sh | sudo bash
   ```

   (Optional) To download the script before running it:

   ```
   curl -O https://packages.corelight.com/install/repositories/corelight/stable/
   ↪script.deb.sh
   sudo chmod +x script.deb.sh
   sudo ./script.deb.sh
   ```

#### Bash script - rpm Installation

1. Run the script using:

   ```
   curl -s https://packages.corelight.com/install/repositories/corelight/stable/
   ↪script.rpm.sh | sudo bash
   ```

   (Optional) To download the script before running it:

   ```
   curl -O https://packages.corelight.com/install/repositories/corelight/stable/
   ↪script.rpm.sh
   sudo chmod +x script.rpm.sh
   sudo ./script.rpm.sh
   ```

#### Manual deb Installation

1. Refresh the package cache:

   ```
   sudo apt-get update
   ```

2. If you are running Debian, install `debian-archive-keyring` so that official Debian repositories are verified. Ubuntu installations can skip this step.

   ```
   sudo apt-get install debian-archive-keyring
   ```

3. Ensure the required tools (curl, gpg, apt-transport-https) are installed before proceeding:

   ```
   sudo apt-get install curl gnupg apt-transport-https
   ```

4. To install a deb repo, you need to install the GPG key that is used to sign repository metadata. Use a utility called apt-key.

   ```
   curl -L https://packages.corelight.com/corelight/stable/gpgkey | sudo apt-key add␣
   ↪-
   ```

5. Verify the file named `/etc/apt/sources.list.d/corelight_stable.list` contains the repository configuration below.

   In the example below, check that the strings **ubuntu** and **trusty** represent your Linux distribution and version:

```
deb https://packages.corelight.com/corelight/stable/ubuntu/ trusty main
deb-src https://packages.corelight.com/corelight/stable/ubuntu/ trusty main
```

Valid options for os and dist parameters can be found in Packagecloud's supported OS list.

6. Update the local APT cache:

```
sudo apt-get update
```

### Manual rpm Installation

1. Install `pygpgme`, a package that allows yum to handle gpg signatures, and a package called `yum-utils` that contains the tools you need for installing source RPMs.

```
sudo yum install pygpgme yum-utils
```

You might need to install the EPEL repository for your system to install these packages. If you do not install `pygpgme`, GPG verification will not work.

2. Create a file named `/etc/yum.repos.d/corelight_stable.repo` that contains the repository configuration below. Replace **el** and **6** in the `baseurl=` path with your Linux distribution and version. Valid options for os and dist parameters can be found in the supported OS list in the docs.

```
[corelight_stable]
name=corelight_stable
baseurl=https://packages.corelight.com/corelight/stable/el/6/$basearch
repo_gpgcheck=1
gpgcheck=0
enabled=1
gpgkey=https://packages.corelight.com/corelight/stable/gpgkey
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300

[corelight_stable-source]
name=corelight_stable-source
baseurl=https://packages.corelight.com/corelight/stable/el/6/SRPMS
repo_gpgcheck=1
gpgcheck=0
enabled=1
gpgkey=https://packages.corelight.com/corelight/stable/gpgkey
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300
```

3. Update the local yum cache:

```
sudo yum -q makecache -y --disablerepo='*' --enablerepo='corelight_stable'
```

### 2.2.2 2. Install Corelight-update

#### Debian

```
sudo apt install corelight-update
```

#### RHEL

```
sudo yum install corelight-update
```

### 2.2.3 3. Add the corelight-update group to existing users (optional)

As part of the installation, a system user and group are added to the OS to manage the Corelight-update service. All files and directories that are created for Corelight-update will belong to the user `corelight-update`, and the group `corelight-update`.

To avoid using `sudo` when running `corelight-update` commands, you can add your user to the `corelight-update` group. For example, use this command to append the `corelight-update` group to the users assigned groups:

```
sudo usermod -aG corelight-update <username>
```

**Tip:** Changes made with the `usermod` command do not take effect in the current session. Logging out and in again will apply the changes.

### 2.2.4 4. Configure Corelight-update (optional)

The Corelight-update utility uses a configuration database to track and maintain the sensor inventory, the global service configuration and interval, the policy assignments, and the integrations.

To configure Corelight-update, start with the sample file as a template, and fill-in the various configuration options. Once you've completed filling in your configuration file, use the file to update the policy defined in the Corelight-update configuration database.

A default policy named **defaultPolicy** is created automatically as part of the installation process. The default policy is functional, but also optional, and can be replaced with custom named policies. There is a configuration sample file provided with the default policy: `/etc/corelight-update/configs/defaultPolicy/db-config.yaml`.

**Attention:** To use Corelight-update for Fleet-managed sensors, you must associate the Fleet Manager sensor policy or suricata policy name with the Corelight-update policy. See **4.1. Change the policy name (optional)** below for the options available.

#### 2.2.4.1 4.1 Change the policy name (optional)

For Fleet-managed sensors, Corelight-update will use the Fleet Manager sensor policy name to collect a group of sensors that it should deploy “Intel” and “Input” content to. Suricata rulesets and package bundles are uploaded to Fleet Manager policies directly.

As of **Fleet Manager v27.8**, Fleet Manager uses separate policies for sensors and Suricata rulesets.

Corelight-update provides a couple options to associate the Corelight-update policy to a Fleet Manager policies:

1. Use the same name for the Corelight-update policy, the Fleet Manager sensor policy and Fleet Manager suricata policy.
2. When adding your Fleet Manager details to the Corelight-update policy inventory, provide the Fleet Manager sensor policy name in the Corelight-update `sensor_policy:` field, and the Fleet Manager suricata policy name in the Corelight-update `suricata_policy:` field. See **4.2 Customize a policy (optional)** below.
3. Change the Corelight-update policy name to match the existing Fleet Manager sensor and suricata policy name (assuming they have the same name).

   For example, to remove the default Corelight-update policy, and create a new policy named “myFleet-Policy”:

   1. Remove the default policy:

   ```
   corelight-update remove -policies defaultPolicy
   ```

   2. Add a new policy named “myFleetPolicy”:

   ```
   corelight-update add -policies "myFleetPolicy"
   ```

   3. Verify the policy is defined:

   ```
   corelight-update show -policies
   ```

   Once the new policy is created, a configuration sample file is created, and placed into a folder with the policy name. For example: `/etc/corelight-update/configs/myFleetPolicy/db-config.yaml`.

**Note:** If the `sensor_policy` field is left blank or the `suricata_policy` field is left blank, it will use the name of the Corelight-update policy for the empty field.

#### 2.2.4.2 4.2 Customize a policy (optional)

Use a configuration example file to create a customized configuration for Corelight-update that defines the Fleet Manager details, sensor inventory, the Corelight-update service configuration, the content assignments, and integrations.

To view a sample policy configuration yaml that enables and pushes the default ETOpen and Corelight suricata rulesets to a single, unmanaged sensor, see *Default policy sources*.

1. Add your sensors to the Corelight-update policy inventory. For information on configuring the sensor inventory, and the use of encrypted passwords, see *Policy inventory settings*.
2. Configure the content you want to deploy to your sensors in the Corelight-update policy configuration. This content can include threat intel, Zeek input files and package bundles, and Suricata rulesets. See *Policy sources*.
3. Configure a custom third-party integration that provides support for a vendor-specific threat source, including source-based customizations and authentication. See *Third-party integrations settings*.

#### 2.2.4.3 4.3 Customize global settings (optional)

1. Customize the default interval settings for data downloading and processing, enabling global integrations, modifying the web service, or deploying global Suricata configuration files. See *Configuration settings*.
2. Customize the default network communication between Corelight-update and Fleet Manager or sensors, including how much time to wait for a status of a file upload. See *Network settings*.

### 2.2.5 5. Add proxy configuration (optional)

See *Using a proxy with Corelight-update* for instructions.

### 2.2.6 6. Run Corelight-update

Run `corelight-update` using the CLI commands, or enable the service. See *Corelight-update Service*.

For additional `corelight-update` command options, see *CLI commands*.

CHAPTER

THREE

# QUICKSTART - UPGRADE

The Corelight-update utility can run as a service at a scheduled intervals, if enabled, to check for updates to the configured data sources, and distribute updated content to the Fleet Manager policies and sensors.

**Attention:** On completion of the upgrade, if you have pre-1.0 release policy files, they must be manually imported into the configuration database. See *CLI commands* for details on the `import` command.

## 3.1 System requirements

For the latest system requirements, see *System requirements* in the references.

## 3.2 Upgrade overview

Set up the Corelight package repository on the host OS if required. See *QuickStart - new install* for instructions.

*1. Upgrade Corelight-update*

*2. (Optional) Configure new Corelight-update features.*

*3. (Optional) For customers upgrading from a version prior to 1.0 or haven't done so previously, add the "corelight-update" group to existing users.*

*4. (Optional) For customers upgrading from a version prior to 1.0, manually import your existing configurations.*

### 3.2.1 Upgrade corelight-update

#### Debian

```
sudo apt update
sudo apt install corelight-update
```

#### RHEL

```
sudo yum install corelight-update
```

### 3.2.2 Configure new Corelight-update features (optional)

When Corelight-update gets upgraded, any existing database will automatically be upgraded.

Use the CLI command `corelight-update show -policy <policy name>` to identify new configuration options or `corelight-update show -policy <policy name> -file /etc/corelight-update/config/<policy name>/db-config.yaml` to replace the existing file with the new format. Then modify as necessary.

Once you've completed filling in your configuration file, use the file to update the policy defined in the Corelight-update configuration database.

When updating policies, you can either supply an entire policy configuration or only the sections you want to update.

**Warning:** When updating from a full or partial configuration, any config section provided must have all none-zero fields provided. Any missing fields will be automatically configured to their **zero** value.

### 3.2.3 Import policy files from versions prior to v1.0 (optional)

The Corelight-update utility uses a configuration database to track and maintain the sensor inventory, the global service configuration and interval, the policy assignments, and any integrations.

If you have pre-1.0 release policy files, they must be manually imported into the configuration database.

You can import your pre-1.0 policies using `corelight-update import` with the `-v0.23` flag to indicate you are importing from a version 0.23 policy. After importing a pre-v1.0 policy, use the `update` command to add the inventory details to the policy. For example, `corelight-update update -policy defaultPolicy -file /etc/corelight-update/configs/defaultPolicy/inventory.yaml`

Once the pre-v1.0 policy is imported, review the imported configuration using the `corelight-update show` command. For example: `corelight-update show -policy defaultPolicy -yaml`

**Note:** The `-v0.23` flag can be used with policies from older versions of corelight-update, but you should always review the imported configuration using the `show` command.

Once a policy has been imported, you will switch to using the new policy configuration to update those policies. The pre-1.0 policy files cannot be used to update a policy, they can only be used as an import.

**Attention:** When Corelight-update gets installed for the first time, it will automatically create the database, a default Global configuration, and a default policy named "defaultPolicy".

If the `corelight-update.db` is deleted, a new `corelight-update.db` will be created the next time the service runs, with a default Global configuration. However, no default policies are created.

CHAPTER

FOUR

# GLOBAL CONFIGURATION

Corelight-update uses a database store global-level and policy-level configurations and settings. Global settings include:

## 4.1 Corelight-update service settings

### 4.1.1 Web service

The web service provides local web access to the documentation, and all of the content created and managed by Corelight-update. The web service is enabled by default, and is optional.

```
webserver:
  enable:   true
  tls:      true
  tls_cert: "/etc/corelight-update/global/cert.crt"
  tls_key:  "/etc/corelight-update/global/cert.key"
  port:     8443
```

**Note:** Updating the default certificate is recommended.

### 4.1.2 Service interval

In some cases it is useful to disable the processing feeds and only have the web service enabled, or modify the default interval for processing data feeds.

```
process_feeds:    true
interval_minutes: "60"
```

When this interval is triggered,

- All caches are updated.
- All local data sources are copied to their respective working folders.
- All remote data sources are copied to their respective working folders.

Additionally, the individual state history for each enabled integration is checked each interval.

- If the integration interval time has lapsed, it processes the integration.
- If the interval has not lapsed, the integration is skipped until the next cycle.

- If the interval is set to `0`, the integrations will be processed each cycle.

For more details, see *Order of operations*

For details on updating the web service or service interval, see *Updating the Global configuration*

**Attention:** The web service and service interval only apply when running Corelight-update as a service. Any changes to these settings require a service restart to take affect.

## 4.2 Configuration settings

### 4.2.1 General settings

#### 4.2.1.1 Additional logging options

If additional logging detail is needed, enable verbose logging. This setting is in addition to the CLI debugging option.

```
verbose: false
```

#### 4.2.1.2 Experimental features

There are currently no experimental features available in Corelight-update.

```
exp_features: false
```

#### 4.2.1.3 Auto-updating policy settings

You can configure Corelight-update to automatically update a policy using a pre-selected configuration file name, and directory path. When `auto_updating_policies` is enabled, Corelight-update monitors the directory path `/etc/corelight-update/configs/<policy_name>/` for a file as defined in `filename`.

On each service interval, Corelight-update checks each path for a policy configuration file, and applies that configuration to the policy. This setting is enabled by default.

```
auto_update_policies:
  enable:   true
  filename: db-config.yaml
```

**Note:** When `auto_updating_policies` is enabled, a configuration file matching the filename setting is **required** in each policy directory: `/etc/corelight-update/configs/<policy_name>/`

#### 4.2.1.4 Pushing content to sensors in parallel

By default, Corelight-update will deploy content updates to the sensors concurrently. Corelight-update will open a connection to multiple sensors in a policy, push updated content, and cycle to the next sensor, up to the `parallel_push_limit` setting.

Content updates are performed in a specified order. To review the order of operations, see *Push content for policies*. The default for `parallel_push_limit` is 10 sensors.

```
parallel_push_limit: 10
```

### 4.2.2 Global-level data sources

Corelight-update supports applying a limited selection of data sources at the Global level.

#### 4.2.2.1 GeoIP database

Enables downloading of the Maxmind GeoIP database. The default interval is 1 week.

```
geoip:
  enable_maxmind: false
  interval_hours: 168
  account_id: 0
  license_key: ""
  database_directory: "/var/corelight-update/files/all/geoip"
```

For additional details, see *Maxmind GeoIP*.

#### 4.2.2.2 Remote data sources

Remote sources are required to be added to each policy configuration. However, any source that's cached globally, will only be downloaded once. See *Remote source settings* for details.

#### 4.2.2.3 Locally managed data sources

In addition to downloading content from external sources for your sensors, Corelight-update will also accept locally-sourced content and configurations that can be applied at a Global-level, or at a Policy-level.

Corelight-update provides folders for input, threat intel, and suricata data at the Global-level and Policy-level, where you can place pre-formatted content to be processed. The following is a list of folder locations files can be placed for automatic processing.

```
/etc/corelight-update/global/global-input
/etc/corelight-update/global/global-intel
/etc/corelight-update/global/global-suricata
/etc/corelight-update/configs/<policy_name>/local-input
/etc/corelight-update/configs/<policy_name>/local-intel
/etc/corelight-update/configs/<policy_name>/local-suricata
```

For example, if an intel file is placed in the `global-intel` folder, the contents are added to the published intel file for all policies. If an intel file is placed in a policy `local-intel` folder, the contents are automatically added to the published intel file only for that policy.

The following functions do not require any additional configuration:

#### Local Intel folders

- All Zeek compatible formatted files in the `global-intel` folder are added to all policies as an intel file.
- Any Zeek compatible formatted files placed in a `local-intel` folder is added to that policy as an intel file.
- Any intel files in the `global-intel`, `local-intel`, or generated by an enabled integration are automatically merged into a single intel.dat file.

#### Local Suricata folders

- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset files placed in the `global-suricata` folder are available to all policies.
- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset placed in a `local-suricata` folder are available to that policy.
- Any ruleset file in the `global-suricata`, `local-suricata`, or generated by an enabled integration are automatically processed and merged into a single suricata.rules file.

#### Local Input folders

- Any Zeek compatible formatted files placed in the `global-input` folder are available to all policies.
- Any Zeek compatible formatted files placed in a `local-input` folder are available to that policy.
- Any input files in the `global-input`, `local-input`, or generated by an enabled integration, (with the same name) will automatically get merged into a single input file with that name.

To review the order that the configurations are processed in, see *Order of operations*.

### 4.2.3 Global-level Suricata settings

If you maintain a centralized set of Suricata configuration files for ruleset tuning and management, you can configure Corelight-update to automatically download your Suricata configuration files from a remote source, and apply them to the Corelight-update connected sensors.

The Suricata configuration files `disable.conf`, `enable.conf` and `modify.conf` can be applied at a global level, and at a policy level. If a `disable.conf`, `enable.conf` or `modify.conf` exist in the Global config directory, they will be processed for each policy automatically.

- To learn about the processing order, see *Order of operations*.
- For information about applying Suricata configuration files at the policy level, see *Suricata policy settings*.

Each time the Corelight-update service runs, the Suricata rulesets can be processed up to three times for each policy:

1. Process any enabled Corelight recommended configs,
2. Process any enabled global-level configs,
3. Process the Suricata policy-level configs.

For example, to pull a `modify.conf` file from GitHub and apply it as part of your Global policy:

```
    remote_global_conf_files:
     - name: modify.conf
       url: https://raw.githubusercontent.com/fakeuser/conf/main/modify.conf
       auth_type: basic
       username: fakeuser
       encrypted_pass: 8946af417b8c3a13358ac42e6f6fbb3f256e2f5cc778a08...
```

The supported authentication types are no auth, `basic`, or `token`. When using the no auth option, leave the `auth_type` field empty.

See *Using a proxy with Corelight-update* for details about using a proxy to download remote sources.

### 4.2.4 Updating the Global configuration

Changes can be made to the global policy using either:

- A config file.
- The Corelight-update CLI command by using the `--global-settings` switch.

#### Updating via `--global-settings`

The Corelight-update CLI command supports updating the Global Configuration directly using the `--global-settings` switch.

- Multiple settings can be updated using a single command.
- Update nested settings by using a "." , for example, `webserver.enable=true`.
- Other than `remote_global_conf_files`, any setting can be updated using a key=value pair.

For example:

```
corelight-update update --global-settings verbose=false interval_minutes=30
```

**Note:** Making changes to a policy using the CLI bypasses the configuration files. To maintain a copy of the current Global Configuration as a config file, export it to a file. See "Show Options" in the *CLI commands*.

See the Complete Global Settings below for a list of fields that can be updated directly.

#### Updating with a config file

When using a config file, additions or changes to a policy are made to a configuration file first before being loaded into Corelight-update to take effect.

To update the global configuration:

1. Output the current global configuration as a file. For example, to create a global config file in yaml format:

   ```
   corelight-update show -global -file /etc/corelight-update/global/config.
   ↪yaml
   ```

2. Change the settings in the config file.
3. Update the global configuration. For example:

```
corelight-update update -global --file /etc/corelight-update/global/
↪config.yaml
```

**Warning:** When making changes to a policy, the configuration file section being modified must also incude any previously defined, non-zero fields. Any fields left undefined will be automatically configured to their **zero** value.

After a configuration has been updated, it's always recommended to verify the global configuration on the console. For example:

```
corelight-update show -global
```

### 4.2.5 Complete global settings

```
verbose: false
exp_features: false
webserver:
    enable: true
    tls: true
    tls_cert: /etc/corelight-update/global/cert.crt
    tls_key: /etc/corelight-update/global/cert.key
    port: 8443
process_feeds: true
interval_minutes: 60
geoip:
    enabled: false
    interval_hours: 168
    account_id: 0
    license_key: ""
    database_directory: /var/corelight-update/files/all/geoip
remote_global_conf_files:
    - name:
      url:
      auth_type:
      auth_token:
      auth_token_header:
      username:
      encrypted_pass:
      ignore_tls:
parallel_push_limit: 10
auto_update_policies:
    enable: true
    filename: db-config.yaml
```

## 4.3 Network settings

Use the global network configuration to modify the communication variables between Corelight-update and the sensors, or a Fleet Manager instance.

### 4.3.1 Updating the Global network config

Changes can be only made to the global network configuration using the Corelight-update CLI command with the `--network-settings` flag.

#### Updating via `--network-settings`

The Corelight-update CLI command supports updating the Global Network Configuration directly using the `--network-settings` flag.

- Multiple settings can be updated using a single command.
- Update nested settings by using a ".". For example, `sensor_timeout_settings.tls_handshake_seconds=10`.
- Any setting can be updated using a key=value pair.

For example:

```
corelight-update update --network-settings sensor_timeout_settings.tls_handshake_
↪seconds=10 sensor_timeout_settings.upload_wait_seconds=60
```

See Complete Global Settings below for a list of fields that can be updated directly.

After a configuration has been updated, verify the global network configuration using the console. For example:

```
corelight-update show -network
```

### 4.3.2 Complete global network settings

```
sensor_timeout_settings:
    transport_dialer_seconds: 5
    tls_handshake_seconds: 10
    idle_conn_seconds: 90
    expect_continue_seconds: 1
    http_seconds: 60
    upload_wait_seconds: 60      # The amount of time to wait for the status of␣
↪an upload to Fleet Manager or a sensor.  In general, errors return within a few␣
↪seconds.
```

CHAPTER

FIVE

# POLICY CONFIGURATION

The Corelight-update policies controls what content is collected, processed, and pushed to Fleet Manager policies and or sensors. This includes third-party integrations, Suricata rules management, Intel file management, Input file management, historical file retention and Zeek package management.

## 5.1 Policy sources

Policy sources represent collections of local and remote pre-formatted data. This includes Suricata rulesets, Intelligence Threat feeds, and other relevant data that can be use with the Input framework.

Corelight-update collects data from these sources, along with data from *third-party integrations*, to be processed according to the respective management settings. For more details, see:

> *Suricata configuration*
>
> *Intel management*
>
> *Input management*

A policy data source differs from third-party integration, as policy data sources must be pre-formatted content you can download using an unauthenticated, basic-authenticated, or token-authenticated URL.

The URL for a remote policy source must be accessible via HTTPS or HTTP. No other protocols are supported.

See *Using a proxy with Corelight-update* for details about using a proxy to download remote sources.

### 5.1.1 Locally managed sources

In addition to downloading content from external sources for your sensors, Corelight-update will also accept locally-sourced content and configurations that can be applied at a Global-level, or at a Policy-level.

Corelight-update provides folders for input, threat intel, and suricata data at the Global-level and Policy-level, where you can place pre-formatted content to be processed. The following is a list of folder locations files can be placed for automatic processing.

```
/etc/corelight-update/global/global-input
/etc/corelight-update/global/global-intel
/etc/corelight-update/global/global-suricata
/etc/corelight-update/configs/<policy_name>/local-input
/etc/corelight-update/configs/<policy_name>/local-intel
/etc/corelight-update/configs/<policy_name>/local-suricata
```

For example, if an intel file is placed in the `global-intel` folder, the contents are added to the published intel file for all policies. If an intel file is placed in a policy `local-intel` folder, the contents are automatically added to the published intel file only for that policy.

The following functions do not require any additional configuration:

#### 5.1.1.1 Local Intel folders

- All Zeek compatible formatted files in the `global-intel` folder are added to all policies as an intel file.
- Any Zeek compatible formatted files placed in a `local-intel` folder is added to that policy as an intel file.
- Any intel files in the `global-intel`, `local-intel`, or generated by an enabled integration are automatically merged into a single intel.dat file.

#### 5.1.1.2 Local Suricata folders

- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset files placed in the `global-suricata` folder are available to all policies.
- Any Suricata formatted ".rules" or ".rules.tar.gz" ruleset placed in a `local-suricata` folder are available to that policy.
- Any ruleset file in the `global-suricata`, `local-suricata`, or generated by an enabled integration are automatically processed and merged into a single suricata.rules file.

#### 5.1.1.3 Local Input folders

- Any Zeek compatible formatted files placed in the `global-input` folder are available to all policies.
- Any Zeek compatible formatted files placed in a `local-input` folder are available to that policy.
- Any input files in the `global-input`, `local-input`, or generated by an enabled integration, (with the same name) will automatically get merged into a single input file with that name.

To review the order that the configurations are processed in, see *Order of operations*.

### 5.1.2 Remote source settings

The following fields are available for configuring a remote policy source:

```
sources:
  - name:
    url:
    source_type:
    global_cache:
    auth_type:
    auth_token:
    auth_token_header:
    username:
    encrypted_pass:
    ignore_tls:
```

- The policy source `source_type` field can be set to either `suricata`, `intel`, or `input`. When using the `intel` or `input` source type, the URL must provide the data in a Zeek compatible format. For `suricata`, the URL must provide the data in the Suricata rule format.

- The `global_cache` is disabled (false) by default for all sources. If `global_cache` is disabled, that source will be download once for each policy that uses it.
- The `auth_type` field can be set to `basic`, `token`, or left empty for `no auth`.

### 5.1.3 Overview of adding policy sources

1. Determine the access url and authentication required for the policy data source.
2. For basic-authenticated sources, use the CLI command `corelight-update encrypt <password>` to generate an encrypted password to store in the policy configuration.
3. Configure the policy data source settings under the `sources:` section of the Corelight-update `db-config` file.

### 5.1.4 Processing a policy source

When Corelight-update processes a policy source, it:

1. Checks the global cache for the target filename.
   1. If the file is present, use the file to process the source.
   2. If the file is not present in the global cache:
      1. Check for a policy level cache of the file and generate an `If-Modified-Since` HTTP header.
      2. Attempt to download the file using the `If-Modified-Since` HTTP header.
         - If a new file is downloaded, create or update the policy-level cache.
      3. Use the policy-level cache to process the source.

**Caution:** Matches are made in the global cache using only the filename, not the full URL.

### 5.1.5 Default policy sources

The default policy provided for Corelight-update includes the pre-configured Suricata rulesets:

- Corelight Labs Suricata Rules
- ET/Open ruleset:

You will find the following pre-configured policy source settings in the `db-config` example file:

```
sources:
  - name: "Corelight"
    url: "https://feed.corelight.com/corelight.rules"
    source_type: "suricata"
    global_cache: true
  - name: "ET/Open"
    url: "https://rules.emergingthreats.net/open/suricata-6.0/emerging.rules.
↪tar.gz"
    source_type: "suricata"
    global_cache: false
```

For more source ideas, see *Commonly used Suricata rulesets*

#### 5.1.5.1 Threat intelligence source example

The threat intel sources managed with Corelight-update must provide their data in a Zeek compatible format.

The following example includes settings for the ThreatQ and MISP threat intel sources:

```
sources:
  - name: ThreatQ
    url: https://string.experience.threatq.online/api/export/
↪c8299290f2d4319923e2eb/?token=aasTjqMXwJ4u
    source_type: intel
    global_cache: false
  - name: MISP
    url: https://misp/attributes/bro/download/all
    source_type: intel
    global_cache: false
    auth_type: token
    auth_token: BVkgNaFh27IGelkIuEAiPBBlDsOp9cjd
    auth_token_header: Authorization
    ignore_tls: true
```

**Note:** For more details on these examples, see *Third-party configuration guides*.

### 5.1.6 Third-party integrations

A third-party integration is a data source that might require custom formatting or parsing of the data for use with a sensor, or has unique authentication requirements.

For more details, and a complete list of the current integrations and their respective configurations, see *Third-party integrations settings*

## 5.2 Policy inventory settings

### 5.2.1 Push content settings

You can use Corelight-update to push content to Corelight Sensors. It supports both Fleet-managed and stand-alone sensors. To push content to sensors, it must be enabled in a policy. Pushing content is disabled by default.

Once pushing content is enabled at the policy level, it can be overridden for non-Fleet-managed at the individual sensor level in the inventory for that policy. See *Inventory settings* below for details.

The policy settings for pushing content are:

```
# Push Content to Sensors
push_content:
    intel:            false
    input:            false
    package_bundle:   false
    suricata:         false
    suricata_configs: false
```

**Tip:** Force Pushing Content

By default, Corelight-update will only push new content to sensors. If you add a sensor to the policy, no content is pushed to it until new content is generated. You can use the CLI to force push existing content to sensors. See *CLI commands* for details.

The policy inventory can include Fleet Manager details and/or a list of the Corelight sensors to deploy content to using Corelight-update. The sensors can be a combination of appliances, such as the hardware, virtual, and software sensors.

Sensors that are Fleet managed do not need to be listed individually in the Corelight-update inventory. Corelight-update will utilize Fleet Manager to deploy content to those sensors. For sensors that aren't Fleet managed, you can push content directly to them by listing their details in the inventory.

If you have version 1.x software sensors, you can use Corelight-update to either push content to the software sensor, or publish threat intel content using Corelight-update's web interface for the software sensor to fetch.

### 5.2.2 Overview of adding Fleet Manager and sensor details to the inventory

1. Prepare a list of the sensors that Corelight-update will deploy to.
   - For Fleet-managed sensors, the sensor inventory will be collected from Fleet Manager.
   - For all standalone appliance sensors: collect the IP address or FQDN, and the sensor username and password.
   - For all version 1.x software sensors: collect the IP address or FQDN, and the host ssh key, or the sensor username and password.
2. If you have Fleet-managed sensors, configure the connection to your Fleet Manager instance under the `fleet:` section of the Corelight-update `db-config` file.
3. Configure the inventory settings under the `sensors:` portion of the Corelight-update configuration file, adding a new `-name` inventory section and associated fields for each non-Fleet managed sensor type in your inventory.
4. Use the configuration file to update the policy in Corelight-update.

### 5.2.3 Inventory settings

The following fields are available for configuring the inventory:

```
fleet:
  ip:                  # fleet address or fqdn
  username:            # fleet username
  password:            # fleet password, leave blank to use encrypted password
  encrypted_pass:      # use the 'encrypt' CLI command to encrypt a password before it's␣
↪stored here
  ignore_tls:          true
  sensor_policy:       # the name of the sensor policy in Fleet Manager (required if␣
↪different from the Corelight-update policy)
  suricata_policy:     # the name of the suricata policy in Fleet Manager (required if␣
↪different from the Corelight-update policy)
sensors:
  - name:                # sensor name
    type:                # physical, virtual, software or localhost
    fleet:               false # true or false
    ip:                  # address or fqdn
    username:            # sensor username
    password:            # set to "ssh-key" to use ssh keys with softsensor, leave blank␣
↪to use encrypted password
```

(continues on next page)

(continued from previous page)

```
    encrypted_pass:  # use the 'encrypt' CLI command to encrypt a password before it
↪'s stored here
    ignore_tls:       true  # physical and virtual sensors ONLY
    suricata:         true  # push suricata rulesets to this sensor
    intel:            true  # push intel files to this sensor
    input:            true  # push input files to this sensor
    bundle:           true  # push package bundle to this sensor
    intel_path:            "/etc/corelight/intel/intel.dat"      # software sensors␣
↪and localhost ONLY
    input_path:            "/etc/corelight/input_files/"         # software sensors␣
↪and localhost ONLY
    suricata_path:         "/etc/corelight/rules/suricata.rules" # software sensors␣
↪and localhost ONLY
    suricata_config_path: "/etc/corelight/suricata/"             # software sensors␣
↪and localhost ONLY
    bundle_path:           "/etc/corelight/corelight.bundle"     # software sensors␣
↪ONLY
```

**Warning:** The **suricata_config_path** `/etc/corelight/suricata/` does not exist by default on Software Sensors v1.x and must be created before Suricata config files can be pushed. The folder must be writable for the username listed for that sensor.

**Encrypted Passwords**

Fleet and individual sensor passwords should be encrypted before they are stored in inventory. Using the `encrypted_pass` field allows you to replace the use of plain text passwords in your Corelight-update configuration file. See **Administering encrypted passwords** later in this topic.

### Add Fleet-managed sensors

Corelight-update can use your Fleet Manager instance to collect an inventory of connected sensors, and deploy content to those sensors.

When Corelight-update is deploying content to Fleet-managed sensors, it uses the Fleet Manager API to authenticate and proxy intel files and input files to those sensors through the Fleet Manager instance. If a Fleet-managed sensor is disconnected from Fleet Manager during the content push, that sensor will not receive files until the next content push (assuming it is connected during the push).

**Suricata rulesets and package bundles are uploaded directly to Fleet Manager and then,**

- **For Fleet Manager versions prior to v27.8**:
  - The **sensor policy** in Fleet Manager is updated to use the new content.
- **Starting with Fleet Manager version v27.8**:
  - The **suricata policy** in Fleet Manager is updated to use the new Suricata ruleset
  - The **sensor policy** is updated to use the new package bundle

Once updated, Fleet Manager will handle pushing the new Suricata ruleset and package bundles to the connected sensors. If Fleet Manager details are configured in the Corelight-update policy, new Suricata rulesets and package bundles will be uploaded even if no sensors are connected to that policy in Fleet Manager.

**To configure Corelight-update to deploy to Fleet-managed sensors, you'll require:**

- Network connectivity from the Corelight-update host to the Fleet Manager. No additional network configuration is required other than the default sensor-to-Fleet communications.
- If the sensor policy name used in Fleet Manager and the policy name in Corelight-update do not match, the sensor policy name must be specified.
- The IP address or FQDN of the Fleet Manager.
- The Fleet username and password.

To enable Corelight-update to communicate with the Fleet instance, configure the `fleet:` section of the configuration file.

```
fleet:
  ip:                 # fleet address or fqdn
  username:           # fleet username
  password:           # fleet password, leave blank to use encrypted password
  encrypted_pass:     # use the 'encrypt' CLI command to encrypt a password before it's␣
↪stored here
  ignore_tls:         true
  sensor_policy:      # the name of the sensor policy in Fleet Manager (required if␣
↪different from the Corelight-update policy)
  suricata_policy:    # the name of the suricata policy in Fleet Manager (required if␣
↪different from the Corelight-update policy)
```

Corelight-update will collect a list of sensors for each Fleet Manager policy automatically. If you have Fleet managed sensors manually configured in the Corelight-update inventory, they can be removed from the inventory, or remain if set to `fleet:  true` in the sensor details. This will cause Corelight-update to skip the sensor while it processes the rest of the policy inventory.

## Add standalone appliance sensors

When Corelight-update is deploying content to appliance sensors, such as the hardware and virtual sensors that are not Fleet-managed, it uses the sensor API to authenticate and deploy content to those sensors.

**To configure a standalone appliance sensor in Corelight-update, you'll require:**

- Network connectivity from the Corelight-update host to the sensor.
- The IP address or FQDN of the sensor.
- The sensor username and password.

The sensor inventory requires one entry for each sensor. You can remove any setting that's not required for a specific sensor's configuration.

```
sensors:
  - name:             # sensor name
    type:             # physical, virtual
    fleet:            false
    ip:               # address or fqdn
    username:         # sensor username
    password:         # leave blank to use encrypted password
    encrypted_pass:   # use the 'encrypt' CLI command to encrypt a password before it
↪'s stored here
    suricata:         true  # push suricata rulesets to this sensor
    intel:            true  # push intel files to this sensor
    input:            true  # push input files to this sensor
    bundle:           true  # push package bundle to this sensor
```

**Fleet Managed Sensors**

If a stanalone appliance sensor is later connected to Fleet Manager, it can be removed from the Corelight-update inventory, or remain if set to `fleet: true` in the sensor details. This will cause Corelight-update to skip the sensor while it processes the rest of the policy inventory.

### Add software sensors

When Corelight-update is deploying content to software sensors, it uses SCP to push updates to a specific folder path on the sensor.

**To configure a software sensor in Corelight-update, you'll require:**

- Network connectivity from the Corelight-update host to the sensor.
- The IP address or FQDN of the sensor.
- The sensor username, and the password or host ssh key.
- The sensor user needs read/write access to the content folders.

**Note:** The command used to reload the suricata rules requires sudo access. If you're deploying Suricata rulesets to a software sensor, the host username will also require passwordless sudo access to apply new rulesets.

The sensor inventory requires one entry for each sensor. You can remove any setting that's not required for a specific sensor's configuration.

```
sensors:
  - name:               # sensor name
    type:               software
    ip:                 # address or fqdn
    username:           # host username
    password:           # set to "ssh-key" to use ssh keys with softsensor, leave blank␣
↪to use encrypted password
    encrypted_pass:     # use the 'encrypt' CLI command to encrypt a password before it
↪'s stored here
    suricata:           true  # push suricata rulesets to this sensor
    intel:              true  # push intel files to this sensor
    input:              true  # push input files to this sensor
    bundle:             true  # push package bundle to this sensor
    intel_path:         "/etc/corelight/intel/intel.dat"      # software sensors and␣
↪localhost ONLY
    input_path:         "/etc/corelight/input_files/"         # software sensors and␣
↪localhost ONLY
    suricata_path:      "/etc/corelight/rules/suricata.rules" # software sensors and␣
↪localhost ONLY
    bundle_path:        "/etc/corelight/corelight.bundle"     # software sensors ONLY
```

### Using Corelight-update to update a sensor on the same host

If Corelight-update is installed on the same host as a software sensor, no connectivity information is required. The only requirement is to include the path on the sensor to place files. Any package bundles will not be moved, they will just get installed.

```
sensors:
  - name:             # sensor name
    type:             localhost
    intel_path:       "/etc/corelight/intel/intel.dat"      # software sensors and␣
↪localhost ONLY
    input_path:       "/etc/corelight/input_files/"         # software sensors and␣
↪localhost ONLY
    suricata_path:    "/etc/corelight/rules/suricata.rules" # software sensors and␣
↪localhost ONLY
    bundle_path:      "/etc/corelight/corelight.bundle"     # software sensors ONLY
```

#### 5.2.3.1 Administering encrypted passwords

Fleet and individual sensor passwords should be encrypted before they are stored in inventory. Using the `encrypted_pass` field allows you to replace the use of plain text passwords in your Corelight-update configuration file.

To use encrypted passwords:

1. Use the Corelight-update CLI command with the in `encrypt` switch to encrypt the password string. When using special characters in your password string, wrap it in quotes. See *CLI commands* for more details.
2. Copy the encrypted password output from the console, and use it to update the `encrypted_pass:` field of the sensor inventory record, or Fleet configuration in the policy configuration file.
3. Verify the `password:` field of the sensor inventory record, or Fleet configuration is empty.
4. Save the changes, and update the Corelight-update policy.

**Note:** A Fleet Manager configuration or sensor inventory record should not have both the `password` and `encrypted_pass` fields populated. Make sure to leave the `password` field blank when using the `encrypted_pass` field. If both fields are populated, the `password` field will be used.

Using the Corelight-update CLI command with the in `encrypt` switch encrypts the password string using AES256 encryption. The encryption master key is randomly generated, and stored in the file `/var/corelight-update/.corelight-update`.

If the master key is removed and regenerated, all encrypted passwords will also have to be regenerated. A password must be encrypted with the current key to be decrypted successfully.

To generate a new master key, delete the existing key, and a new one will automatically be created when needed.

## 5.3 Suricata configuration

In addition to downloading Suricata rulesets from multiple sources, Corelight-update can manage the ruleset. It works by optionally applying Corelight recommended changes to the rulesets, and extracting content from Suricata rules and creating Zeek Intel rules with that content.

Content is only extracted from enabled rules and the “do_notice” flag can individually be set based on rule type. This means you can use the typical enable.conf and disable.conf rules to control what data is extracted. See *Suricata policy settings* for details.

**Tip:** No configuration is required to include local Suricata rulesets. See *Locally managed sources* for details.

- Any “.rules” or “.rules.tar.gz” ruleset placed in the global-suricata folder is automatically available to all policies.
- Any “.rules” or “.rules.tar.gz” ruleset placed in a local-suricata folder is automatically available to that policy.

### 5.3.1 Suricata configuration files

Suricata uses four configuration files when processing traffic and/or testing rules.

- suricata.yaml
- classification.config
- reference.config
- threshold.config

These configuration files can be manually placed in the policy configs folder (`/etc/corelight-update/configs/<policy>/`), or the policy can be configured to pull Suricata configuration files from remote sources if desired. See *Remote config files*.

See *Using a proxy with Corelight-update* for details about using a proxy to download remote sources.

Optionally, these configuration files can be pushed to the policy in Fleet Manager or directly to a sensor. See *Push content settings*.

**Warning:** Suricata configuration files are not pushed to Software Sensor v1.x.

### 5.3.2 Disabled rules

By default, disabled rules are not written back to the final Suricata ruleset. If desired, disabled rules can be included in the ruleset file by enabling `write_disabled_rules: true` in the *Suricata policy settings*.

### 5.3.3 Ruleset testing

By default, Corelight-update attempts to test the ruleset using Suricata, if it's available on the host running Corelight-update. If Suricata is not available, Corelight-update logs that it did not test the ruleset and continues.

If the rulesets is tested, and one or more rules fail the test, the details of the failed rules are logged and processing continues. Optionally, Corelight-update can be configured to discard a failed ruleset, after the failed rules have been logged, by setting `fail_on_ruleset_error: true` in the *Suricata policy settings*.

If any of the Suricata configuration files are placed in the policy configuration folder, or pulled from a remote location, they are automatically used when testing the Suricata ruleset.

**Tip:** It is recommended to use the same version of Suricata for testing that will be used in production. Testing with the Corelight version of Suricata can be accomplished by installing the Corelight Softsensor (without a license) on the same host running Corelight-update.

For debian based installation, Software Sensor is automatically installed as a "recommended" package. This can be disabled by adding the `--no-install-recommends` when installing Corelight-update.

Corelight-update and Software Sensor use the same package repository so the installation only requires a single command. See Software Sensor Online Installation for details.

See the following sections for more details:

#### 5.3.3.1 Suricata policy settings

The configuration options mentioned in *Suricata configuration* can be changed with the following settings:

```
# Suricata ruleset processing
suricata:
  corelight_recommended_disable:     true
  corelight_recommended_enable:      true
  corelight_recommended_modify:      true
  write_disabled_rules:              false
  ip_extraction:                     true
  ip_do_notice:                      true
  ja3_extraction:                    true
  ja3_do_notice:                     true
  test_ruleset:                      true
  fail_on_ruleset_error:             false
  remote_conf_files:
     - name:
       url:
       auth_type:
       auth_token:
       auth_token_header:
       username:
       encrypted_pass:
       ignore_tls: false
```

**Atomic rule extraction**

Currently, only IP and JA3 based rules can be extracted. For IP based rules, the rule has to have a subnet or IP address in the rule. If it only uses a address group, it will not get extracted.

### Remote config files

If you maintain a centralized set of Suricata configuration files for ruleset tuning and management, you can configure Corelight-update to automatically download the files from a remote source, and apply them to the Corelight-update connected sensors.

The Suricata configuration files `disable.conf`, `enable.conf` and `modify.conf` can be applied at a global, and at a policy level.

- To learn about the processing order, see *Order of operations*.
- For information about setting Suricata configuration files at the Global level, see *Configuration settings*.

For example, to pull a `modify.conf` file from GitHub:

```
remote_conf_files:
 - name: modify.conf
   url: https://raw.githubusercontent.com/fakeuser/conf/main/modify.conf
   auth_type: basic
   username: fakeuser
   encrypted_pass: 8946af417b8c3a13358ac42e6f6fbb3f256e2f5cc778a08...
```

The supported authentication types are no auth, `basic`, or `token`. When using the no auth option, leave the `auth_type` field empty.

Supported Suricata configuration files include:

- disable.conf
- enable.conf
- modify.conf
- suricata.yaml
- classification.config
- reference.config
- threshold.config

#### 5.3.3.2 Suricata rules management

Corelight-update uses the familiar disable, enable, and modify.conf files to process and manage Suricata rules. However, Corelight-update offers significant performance and functionality improvements compared to other solutions.

Once all the rules from all the sources are downloaded and merged, Corelight-update makes up to three passes processing the rules:

1. The first pass will process Corelight recommended modifications (if enabled).
2. The second pass will process global modifications.
3. The third pass will process the individual policy modifications.

For each pass, any disable rule filters (disable.conf entries) are processed, then the enable rule filters (enable.conf entries), followed by rule modifiers (modify.conf entries).

### File filters

In addition to disabling individual rules, the `disable.conf` entries can be used to ignore entire rulesets by file name. Filters are used to identify and ignore ruleset files as they are copied to the working directory for processing. After the files are downloaded and uncompressed (as necessary), if a ruleset filename matches an entry in `disable.conf`, it is ignored.

```
# Examples of disabling by file name.
Filename:emerging-icmp.rules
Filename:emerging-dos
Filename:emerging
group:emerging-icmp.rules
group:emerging-dos
group:emerging
```

**Important:** The `Filename` filter matches all file names that begin with the entry.

### Rule filters

To disable a rule that is enabled by default, add the rule to the disable.conf file. To enable a rule that is disabled by default, add the rule to the enable.conf file.

There are multiple methods to identify rules to be disable or enabled. One method, rule filters can be added by listing the Signature ID <SID> or Generator ID:Signature ID combination <GID>:<SID>.

```
# GID:SID
<sid>
<gid>:<sid>
```

Another method is to use regex. Rule filters that use a regex pattern will be applied to rules that match that pattern.

**Note:** Regex patterns must be wrapped in double quotes or have any white space removed. Use a `\s` to represent white space.

Special characters also have to be escaped, for example, use `\$` for `$`.

```
# Regex
re:<regex string>

# Match all rules that begin with "alert udp $HOME_NET any -> any 53"
re:alert\sudp\s\$HOME_NET\sany\s->\sany\s53
re:"alert udp \$HOME_NET any -> any 53"
```

A method unique to Corelight-update, rule filters can also be added individually or in groups with Field:Value pairs. Use any of these fields to identify the rule:

```
# Field:Value
Protocol:<value>
SrcAddress:<value>
SrcPort:<value>
DestAddress:<value>
DestPort:<value>
```

(continues on next page)

(continued from previous page)

```
Classtype:<value>
Metadata:<contains value>
```

When using the Metadata field to identify a rule, if there are any white spaces in the string to look for, it must be wrapped in double quotes.

### Rule modifiers

To modify a Suricata rule, identifying the rule is the same as rule filters, with the exception that multiple rules can also be identified with GID:SID pairs. Multiple GID:SID entries on the same line need to be comma separated.

Rules can be identified and modified one of four ways:

- The legacy format: `<gid:sid> "<from regex>" "<to string>"` (The `gid` is optional.)
- The legacy regex format: `re:<rule regex> "<from regex>" "<to string>"`
- The new Corelight-update regex format: `re:<rule regex> <field>:<value>`
- The new Corelight-update format: `<rule> <field>:<value>`

**Tip:** See the Suricata documentation for more information about Suricata rules format.

### Legacy format and Legacy regex format

The legacy and legacy regex formats require the <from regex> and <to string> statements to be enclosed in double quotes, and separated with a space `"<from regex>" "<to string>"`. The `" "` between the expressions delineates the two.

With the legacy format, the rule identifier is a combination of one or more `GID:SID` combinations. With the legacy regex format, the rule identifier is a regex pattern `re:<rule regex>`. For example,

```
# Legacy format
<sid> "<from regex>" "<to string>"
<gid>:<sid> "<from regex>" "<to string>"
<sid>,<gid>:<sid>,<gid>:<sid> "<from regex>" "<to string>"
<sid>,<sid>,<sid>,<sid> "<from regex>" "<to string>"

# Legacy regex format
re:<rule regex> "<from regex>" "<to string>"
```

**Caution:** Regex patterns used to identify the rule must be wrapped in double quotes or have any white space removed. Use a `\s` to represent white space.

### Corelight-update regex format

The Corelight-update regex format can use a regex pattern to identify the rule or rules to be modified, and then use the new Field:Value method to modify the rule.

In the `Field:Value` pair, values can be enclosed in double quotes but are not required (unless double quotes are required in that signature field i.e. “Msg”)

```
re:<regex string> <field>:<value>

# Match all rules that begin with "alert udp $HOME_NET any -> any 53"
# and change the "Classtype" to "misc-attack"
re:alert\sudp\s\$HOME_NET\sany\s->\sany\s53 Classtype:misc-attack
```

Once the rule is identified, it can be modified by listing the field and the value it should be set to. (See *Modify examples*)

**New Modify Options for v1.3.0**

New in Corelight-update v1.3.0, content can now be appended to the “Metadata” and “Other” fields with `MetadataAppend` and `OtherAppend` respectively. If the “Append” field name is used, any content in the “Value” section will be appended with a space between the existing content and the new content.

```
# Any of the following fields can be modified:
Protocol
SrcAddress
SrcPort
DestAddress
DestPort
Msg
SID
GID
Rev
Classtype
Metadata
MetadataAppend # Any content added with "MetadataAppend" will be appended to the␣
↪existing content.
Threshold
Priority
Target

# All other Option fields have to be modified as a single string with field "Other"
Other   i.e. Other:reference:url,blacklist.3coresec.net/lists/et-open.txt;
OtherAppend # Any content added with "OtherAppend" will be appended to the existing␣
↪content.
```

### Corelight-update format

The same as above, in the `Field:Value` pair, values can be enclosed in double quotes but are not required (unless double quotes are required in that signature field i.e. "Msg")

Any of these fields can be used to identify the rule:

```
# GID:SID
<sid>
<gid>:<sid>
<sid>,<gid>:<sid>,<gid>:<sid>
<sid>,<sid>,<sid>,<sid>

# Field:Value
Protocol:<value>
SrcAddress:<value>
SrcPort:<value>
DestAddress:<value>
DestPort:<value>
Classtype:<value>
Metadata:<value contains>  # The value contains is a string in the metadata field␣
↪used to identify the rule.
```

**Metadata contains**

If the metadata contains value includes white spaces, it must be wrapped in double quotes.

Once the rule is identified, the same fields listed for Corelight-update regex format (listed above) can be modified by listing the field and the value.

### 5.3.3.3 Modify examples

It is common to modify the source and/or destination of a rule. Multiple addresses or ranges of addresses can be assigned to the same rule. See the Suricata documentation for examples of source and destination operators.

This example modifies a rule so that it only matchs on traffic coming from all `$HOME_NET` sources except 192.168.0.1.

```
# disable signature for 192.168.0.1
#alert udp $HOME_NET any -> $EXTERNAL_NET [!3478,1023:] (msg:"ET INFO Session␣
↪Traversal Utilities for NAT (STUN Binding Request On Non-Standard High Port)";
↪ content:"|00 01|"; depth:2; content:"|21 12 a4 42|"; distance:2; within:4;␣
↪reference:url,tools.ietf.org/html/rfc5389; metadata:created_at 2021_06_03, updated_
↪at 2021_06_03; classtype:attempted-user; gid:1; sid:2033078; rev:2;)
2033078 SrcAddress:[$HOME_NET,!192.168.0.1]
```

**Tip:** The unedited rule was added as a comment just to document the original rule.

This example will modifies the rule so it matches any source except 192.168.0.1, and any destination except 192.168.0.2.

```
# disable signature 2031297 for traffic between 192.168.0.1 and 192.168.0.2
2031297 SrcAddress:!192.168.0.1
2031297 DestAddress:!192.168.0.2
```

This example modifies the rule so it matches all customer networks except customer "B".

```
# enable signature 2027397 for all customer except "B" to any destination and␣
↪updating the revision to 4
2027397 SrcAddress:[$All_CUSTOMERS,!$CUSTOMER_B]
2027397 DestAddress:any
2027397 Rev:4
```

The following example modifies the priority of all rules with a classtype of "attempted-user" to 1.

```
# modify the priority of all rules with a classtype of "attempted-user" to 1
Classtype:attempted-user Priority:1
```

The following example modifies all rules with a specific classtype to another classtype.

```
# Change all rule of Classtype "misc-activity" to a Classtype of "cool-activity"
Classtype:misc-activity Classtype:"cool-activity"
```

## 5.4 Intel management

You can leverage the Zeek Intelligence framework to match a list of IOC's against live network traffic on the sensor. Use Corelight-update to validate and merge one or more threat intel files, and publish a single, integrated threat intel file.

### 5.4.1 Intel management settings

```
intel_management:
  intel_file_cleanup: true
  max_intel_file_age: 24
  disable_filename: disable.intel
```

Every time a new intel file is generated, a copy of the file with the current timestamp is also created. The `intel_file_cleanup` and `max_intel_file_age` (in hours) control the retention of the timestamped copies.

### 5.4.2 Disable Threat Intel indicators

If provided, Corelight-update will also use an intel disable file `disable_filename` to remove unwanted indicators from the published intel file, allowing you to effectively "disable" specific threat intel indicators.

The disable.intel file is a text file with a single column of indicators to remove.

```
indicator
<disabled indicator value 1>
<disabled indicator value 2>
```

### 5.4.3 Add Threat Intel sources

Threat intel sources are collections of IOC's in Zeek compatible formatted files. These files can be provided by a variety of sources, including security vendors, and as open source IOC collections.

Corelight-update can pull threat intel sources hosted in local and remote repositories.

To add threat intel sources, you'll configure them as Corelight-update *Policy sources*.

For an example of a third-party Threat Intel policy source configuration, see *Threat intelligence source example*

To review the order that the configurations are processed in, see *Order of operations*.

## 5.5 Input management

You can leverage the Zeek Input framework to provide contextual data for use with enabled Zeek packages. Depending on the Zeek packages, this data can be used to generate logs (alerts), prevent the generation of logs (alerts), and/or enrich logs with additional data from external sources.

Corelight-update can collect input files from local or remote sources and/or generate input files with enabled *third-party integrations*. Once collected, any input files with the same name will automatically get merged into a single input file with that name and published.

### 5.5.1 Input management settings

```
input_management:
  default_input:          true
```

Corelight sensors contain a number of Zeek packages that can take advantage of input files. However, none of those files are included out of the box. If `default_input` is enabled, Corelight-update will automatically generate templates for those files and place them in the local-input folder. See *Locally managed sources* for the path.

## 5.6 Third-party integrations settings

Third-party integrations provide support for a vendor-specific threat source, including source-based customizations and authentication.

Third-party integrations differ from Corelight-update *Policy sources*, in that a Policy source must be pre-formatted content you can download using an unauthenticated, basic-authenticated, or token-authenticated URL.

### 5.6.1 CrowdStrike

The same connection details are used for all four CrowdStrike Integrations, as long as it has the required access.

**Attention:** Downloading Suricata rules from CrowdStrike requires a Falcon Intelligence Premium subscription. The Client ID and Client Secret need access to the following API: *https://api.crowdstrike.com/intel/entities/rules-latest-files/v1*

Downloading intel indicators from CrowdStrike requires a Falcon Intelligence subscription or better. The Client ID and Client Secret need access to the following API: *https://api.crowdstrike.com/intel/combined/indicators/v1*

General CrowdStrike configuration settings:

```
crowdstrike_config:
  id: "<FALCON_CLIENT_ID>"
  secret: "<FALCON_CLIENT_SECRET>"
  member_cid: "<FALCON_MEMBER_CID>"
  cloud: "<FALCON_CLOUD>"            # us-1, us-2, eu-1, us-gov-1
  host_override:
  base_path_override:
  debug:
```

### 5.6.1.1 Falcon Spotlight - CVEs

The CrowdStrike Falcon Spotlight CVE integration will download data about all hosts with CVE's that match the provided criteria. If no CVE "status" or "severity" is specified, all CVE's who's status is *NOT* "closed" will be downloaded.

CrowdStrike Falcon Spotlight relies on endpoint agents, and does not schedule "network scans" to identify vulnerabilities. As a result, frequently downloading data from Falcon Spotlight can provide near-realtime updates. If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, the data will be used to create an Input Framework file that can be used by a Zeek script to generate new logs, or enrich existing logs, such as the **suricata_corelight.log** and **notice.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" is enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

#### Settings

```
crowdstrike_spotlight_cve:
  enabled: true
  interval_hours: 0
  filename: cve_data.tsv
  request_limit: 5000      # max 5000
  status: open,reopen      # comma separated, one or more of: open, reopen, closed,␣
↪expired
  severity: critical       # comma separated, one or more of: critical, high, medium,␣
↪low, unknown, none
```

#### Input file

The following is a sample input file created by this integration, using tab-separated values.

```
#fields ip      hostname  host_uid                          machine_domain    os_
↪version           source                 cve_list
10.21.0.102     DC        fb5946b0422e4da49e4575995fb89060  windomain.local   Windows␣
↪Server 2016  CrowdStrike Spotlight  CVE-2022-26904,CVE-2022-34701,CVE-2020-0911,CVE-
↪2022-24479,CVE-2020-1477
192.168.1.120   fleet     04a15f26ace249f68c583fd7be70f9db  -                 Ubuntu␣
↪20.04         CrowdStrike Spotlight  CVE-2020-12313,CVE-2020-12319,CVE-2022-36402,
↪CVE-2022-38096,CVE-2022-38457
```

(continues on next page)

(continued from previous page)

```
192.168.1.155   ss2oh     62c850ec617843f8959f1442843bb816  -                 Ubuntu␣
↪20.04         CrowdStrike Spotlight  CVE-2020-12313,CVE-2020-12319,CVE-2022-36402,
↪CVE-2022-38096,CVE-2022-38457
192.168.12.212  skynet    fae3f73ce1404e0aae1626dbddfc3fe8  -                 Ubuntu␣
↪22.04         CrowdStrike Spotlight  CVE-2020-12313,CVE-2016-1585,CVE-2022-36227,
↪CVE-2022-45884,CVE-2023-22995,CVE-2022-38457
192.168.56.102  DC        34a6b864b61146d6ad051a9d63a5585f  windomain.local   Windows␣
↪Server 2016  CrowdStrike Spotlight  CVE-2017-11771,CVE-2022-34718,CVE-2019-0736,CVE-
↪2022-35744
192.168.56.103  WEF       c6f3d2351739482baf36cc6e4af65163  windomain.local   Windows␣
↪Server 2016  CrowdStrike Spotlight  CVE-2017-11771,CVE-2022-34718,CVE-2019-0736,CVE-
↪2022-35744
192.168.56.103  WEF       0bb70f50a9a3470dbc3e09bd6eb18fc4  windomain.local   Windows␣
↪Server 2016  CrowdStrike Spotlight  CVE-2022-26904,CVE-2022-34701,CVE-2020-0911,CVE-
↪2022-24479
192.168.56.104  WIN10     a71be784db1a40e5b0fd7e6b73f6c7b7  windomain.local   Windows␣
↪10           CrowdStrike Spotlight  CVE-2021-36965,CVE-2021-43217,CVE-2022-22012,
↪CVE-2020-9633,CVE-2021-24077
192.168.56.104  WIN10     23fac76b0e5246f8b8ba22d1bbd6bc04  windomain.local   Windows␣
↪10           CrowdStrike Spotlight  CVE-2022-23279,CVE-2020-1286,CVE-2021-33784,CVE-
↪2022-23299,CVE-2020-1391
```

**Important:** To use this file, an additional Zeek package is required on the sensor.

## suricata_corelight log

A typical suricata_corelight.log provides content like the example below:

```
  {
    _path: suricata_corelight
    _system_name: Lab-AP200
    _write_ts: 2023-08-01T01:22:20.096550Z
    alert.action: allowed
    alert.category: Attempted Information Leak
    alert.gid: 1
    alert.metadata: [ [-]
      created_at:2023_07_28
      updated_at:2023_07_28
    ]
    alert.rev: 1
    alert.severity: 2
    alert.signature: **CONTROL** curl User-Agent Outbound CVE-2020-12313
    alert.signature_id: 1000000
    community_id: 1:hfVPB4FWl48hOvuIzwyVBvzWBwY=
    flow_id: 11005288195832
    id.orig_h: 192.168.12.212
    id.orig_p: 48086
    id.resp_h: 3.160.22.77
    id.resp_p: 80
    id.vlan: 12
    pcap_cnt: 0
    service: http
    suri_id: Sp5Hxvr0blDf
```

(continues on next page)

(continued from previous page)

```
    ts: 2023-08-01T01:22:20.092276Z
    tx_id: 0
    uid: CUWMCe4TJo8pS41Rnj
  }
```

If the Zeek package is loaded, the suricata_corelight.log will be enriched with additional content provided by the integration, like the example below:

```
  {
    orig_vulnerable_host.cve: CVE-2020-12313
    orig_vulnerable_host.host_uid: fae3f73ce1404e0aae1626dbddfc3fe8
    orig_vulnerable_host.hostname: skynet
    orig_vulnerable_host.os_version: Ubuntu 22.04
    orig_vulnerable_host.source: CrowdStrike Spotlight
  }
```

**Note:** Field names begin with "orig" or "resp" to identify which host is referenced.

### 5.6.1.2 Falcon Spotlight - Hosts

The CrowdStrike Falcon Spotlight Hosts integration will download data about all entity_types that match the provided criteria. If no "entity_type" is specified, all known entities (that have a current IP address) will be listed.

CrowdStrike Falcon Spotlight relies on endpoint agents and does not scheduled "network scans" to identify network entities. As a result, frequently downloading data from Falcon Spotlight can provide near-realtime updates. If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, the data will be used to create an Input Framework file that can be used by a Zeek script to generate new logs, or enrich existing logs, such as the **known_hosts.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" in enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

#### Settings

```
crowdstrike_spotlight_hosts:
  enabled: true
  interval_hours: 0
  entity_type: ""              # managed, unmanaged, or unsupported
  filename: hosts_data.tsv
```

## Input file

The input file contains the following information (if it's available):

- IP address (required)
- MAC address
- Hostname
- Host Unique ID
- OS version
- Endpoint status (required)
- Machine domain
- Additional description
- Endpoint information source (required)

The following is a sample input file created by this integration, using tab-separated values.

```
#fields  ip     mac                 hostname  host_uid                              os_
↪version    status        machine_domain  desc                         source
192.168.56.103  00-50-56-A3-B1-C2  WEF       ced83f0c26493b638086fdc7b8b2c01d     -  ␣
↪           managed       -             Falcon Discover details  CrowdStrike
10.21.0.102     00-50-56-A1-B1-C4  DC        c53fdc3178ba36759c471d6b6655e324     -  ␣
↪           managed       -             Falcon Discover details  CrowdStrike
192.168.56.104  00-50-56-A2-B1-C2  WIN10     abb6c27309cf3730bb73e8cfd732d838    ␣
↪Windows 10    managed       lab.local       Falcon Discover details  CrowdStrike
192.168.1.155   92-91-E0-3E-66-A8  ss2oh     9caa11e26d1f371797e73e9b9199d481     -  ␣
↪           managed       -             Falcon Discover details  CrowdStrike
192.168.1.120   00-0C-29-AB-75-05  fleet     81f845fe72ae32168aba94707fc8a49f     -  ␣
↪           managed       -             Falcon Discover details  CrowdStrike
192.168.12.1    -                  -         613cd0e8a671350e83dec735143db1e0     -  ␣
↪           unsupported   -             Falcon Discover details  CrowdStrike
192.168.12.210  -                  -         5f67453d7e833b0f82ac1d7a5788142a     -  ␣
↪           unmanaged     -             Falcon Discover details  CrowdStrike
192.168.12.222  -                  -         5abcec34b3443f3cb7fe17c4f7100e02     -  ␣
↪           unmanaged     -             Falcon Discover details  CrowdStrike
192.168.12.212  00-50-56-A1-1F-07  skynet    439293445449716808dec735143db1e9    ␣
↪Ubuntu 22.04  managed       -               Falcon Discover details  CrowdStrike
```

**Important:** To use this file, the "zeek-endpoint-enrichment" package is required on the sensor.

## known_hosts log

A typical known_hosts.log provides content like the example below:

```
  {
    _path: known_hosts
    _system_name: Lab-AP200
    _write_ts: 2023-08-22T13:20:59.526107Z
    annotations: [ ]
    conns_closed: 167
```

(continues on next page)

(continued from previous page)

```
    conns_opened: 167
    conns_pending: 0
    duration: 920.0746190547943
    endpoint.desc: Falcon Discover details
    endpoint.host_uid: 439293445449716808dec735143db1e9
    endpoint.os_version: Ubuntu 22.04
    endpoint.source: CrowdStrike
    endpoint.status: managed
    host_ip: 192.168.12.212
    kuid: Kf1THOpT9hJa5
    last_active_interval: 954.522488117218
    last_active_session: KfqhT6kg6fP7k
    long_conns: 0
    ts: 2023-08-22T13:04:54.000617Z
  }
```

### known_devices log

A typical known_devices.log provides content like the example below:

```
  {
    _path: known_devices
    _system_name: Lab-AP200
    _write_ts: 2023-08-22T13:20:59.526107Z
    annotations: [
      CrowdStrike/managed
    ]
    duration: 920.0746190547943
    host_ip: 192.168.12.212
    kuid: Kf1THOpT9hJa5
    last_active_interval: 954.522488117218
    last_active_session: KfqhT6kg6fP7k
    mac: 00:50:56:A1:1F:07
    num_conns: 0
    protocols: [
      CrowdStrike
    ]
    ts: 2023-08-22T13:04:54.000617Z
    vendor_mac: unknown
  }
```

### known_domains log

A typical known_domains.log provides content like the example below:

```
  {
    _path: known_domains
    _system_name: Lab-AP200
    _write_ts: 2023-08-22T13:51:39.591783Z
    annotations: [
      CrowdStrike/managed
    ]
    domain: LAB.LOCAL
```

(continues on next page)

(continued from previous page)

```
    duration: 0
    host_ip: 192.168.56.104
    kuid: KfkAPIKyTuYv3
    last_active_interval: 42202.37188410759
    last_active_session: Kf64KcY1eZwM
    num_conns: 1
    protocols: [
      CrowdStrike
    ]
    ts: 2023-08-22T13:47:23.586163Z
  }
```

#### known_names log

A typical known_names.log provides content like the example below:

```
  {
    _path: known_names
    _system_name: Lab-AP200
    _write_ts: 2023-08-22T13:20:59.526107Z
    annotations: [
      CrowdStrike/managed
    ]
    duration: 920.0746190547943
    host_ip: 192.168.12.212
    hostname: SKYNET
    kuid: Kf1THOpT9hJa5
    last_active_interval: 954.522488117218
    last_active_session: KfqhT6kg6fP7k
    num_conns: 0
    protocols: [
      CrowdStrike
    ]
    ts: 2023-08-22T13:04:54.000617Z
  }
```

### 5.6.1.3 Suricata Ruleset

The CrowdStrike Falcon Suricata ruleset file will only be downloaded if it has changed since the last interval.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, the ruleset will be processed with the rulesets from all other sources.

#### Settings

```
crowdstrike_suricata:
  enabled:                   false
  interval_hours:            0
```

### 5.6.1.4 Falcon Threat Intelligence

The CrowdStrike Falcon Indicators integration will download all requested indicators at each interval.

There are several configurable options for CrowdStrike indicators. Select the malicious confidence level, how many days worth of history, and which indicators to collect.

**Note:** Due to the high number of hash indicators available, the length of history is configured separate from other types of indicators.

Intel Malicious confidence options are: “high”, “medium”, “low”, or “unverified”. The following definitions apply to *malicious_confidence*:

- high: If indicator is an IP or domain, it has been associated with malicious activity within the last 60 days.
- medium: If indicator is an IP or domain, it has been associated with malicious activity within the last 60-120 days.
- low: If indicator is an IP or domain, it has been associated with malicious activity exceeding 120 days.
- unverified: This indicator has not been verified by a CrowdStrike Intelligence analyst or an automated system.

Once downloaded, the data will be merged with all other intel files (if there are any), and published. If “intel” in enabled in the “push_content” settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

#### Settings

```
crowdstrike_indicators:
  enabled:                         false
  interval_hours:                  1
  request_limit:                   50000
  enable_do_notice:                true
  malicious_confidence:            high
  last_updated_days:               60
  hash_last_updated_days:          3
  indicator_type_ip_address:       true
  indicator_type_ip_address_block: true
  indicator_type_url:              true
  indicator_type_https_url:        false
  indicator_type_email_address:    true
  indicator_type_domain:           true
  indicator_type_x509_subject:     true
  indicator_type_username:         true
  indicator_type_hash_md5:         true
  indicator_type_hash_sha256:      false
  indicator_type_file_name:        true
```

(continues on next page)

(continued from previous page)

```
  targets:
  threat_types:
```

**Error:** The default request limit is set to 50,000, which works for most customers. However, for some customer subscriptions the request limit cannot be more than 10,000 or an error is returned.

In addition to configuring which indicators to collect, you can also filter the indicators based on the type of target or the threat type.

- To list a single Target or Threat Type, enter the string with both double quotes and single quotes.
- To list multiple Targets or Threat Types, enter the string with both double quotes and square brackets around the entire string, and single quotes around each item.

Examples:

```
targets:          "'Aerospace'"
threat_types:     "['Commodity','Ransomware']"
```

### Intel log

This integration will enrich the intel.log with content like the following:

```
{
  "@path":"intel",
  "@sensor":"Lab-AP200",
  "@timestamp":"2023-01-06T05:13:38.841292Z",
  "ts":"2023-01-06T05:13:38.841292Z",
  "uid":"CNh51N3dSRfMZG1Pt4",
  "id.orig_h":"195.133.40.86",
  "id.orig_p":64910,
  "id.resp_h":"192.168.13.20",
  "id.resp_p":80,
  "seen.indicator":"77.247.181.165",
  "seen.indicator_type":"Intel::ADDR",
  "seen.where":"Conn::IN_ORIG",
  "matched": [
    "Intel::ADDR"
  ],
  "sources": [
    "blocklist_de",
    "cinsscore_ci_badguys",
    "blocklist_net_ua",
    "Mandiant",
    "dshield_block"
  ],
}
```

**Attention:** The Spotlight integrations require an additional Zeek script to be loaded on the sensors. See *Zeek package management*. If you enable the integration, Corelight-update will upload the input file to the sensor. But if the required script isn't available on the sensor, the input data won't be used.

### 5.6.2 Mandiant Threat Intelligence

Configure the Mandiant Threat Intelligence integration to set how frequently the integration runs, how much history to initially download, how much history to use in an Intel file, and how much history to maintain in the SQLite DB. This integration uses the Mandiant Threat Intelligence API v4. To use the v2 API, see *FireEye iSIGHT Threat Intelligence*.

**do_notice**

The `do_notice` flag can be set based on the individual indicator type, and an overall minimum Confidence Score. For example, setting the `min_confidence_score_doNotice:  95`, would only set the `do_notice` flag to `T`, if the Mandiant Confidence score was 95% or better. It is not set in the database; only when the intel file is created.

**Tip:** By default, only MD5 hash support is enabled on a Corelight Sensor. It is recommended that you use only one hash type. If you plan on using another hash type, update the configuration and enable the appropriate package on the sensor.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

#### 5.6.2.1 Settings

```
mandiant_threat_intel:
  enabled: true
  interval_hours: 0
  url: https://api.intelligence.mandiant.com
  key_id:
  key_secret:
  download_history: 89  # must be less than 90 days
  max_history: 365      # how much history to keep in the local database
  use_history: 25       # how much history to use in the intel file
  debug: false
  min_confidence_score_use: 80          # minimum confidence score to use in the intel␣
↪file
  min_confidence_score_download: 60     # minimum confidence score to download into␣
↪the local database
  min_confidence_score_doNotice: 100    # minimum confidence score to set the do_
↪notice flag
  exclude_os_indicators: false          # exclude open source indicators from the␣
↪download into the local db
  request_limit: 1000
  indicator_type_url: true
  do_notice_url: true         # do_notice flags will only be enabled if 'min_
↪confidence_score_doNotice' is met
  indicator_type_fqdn: true
  do_notice_fqdn: true
  indicator_type_ipv4: true
  do_notice_ipv4: true
  indicator_type_md5: true      # it's recommended to only enable 1 hash indicator␣
↪type (MD5, SHA1 or SHA256)
  do_notice_md5: true
  indicator_type_sha1: false    # it's recommended to only enable 1 hash indicator␣
↪type (MD5, SHA1 or SHA256)
  do_notice_sha1: true
```

(continues on next page)

(continued from previous page)

```
indicator_type_sha256: false  # it's recommended to only enable 1 hash indicator␣
↪type (MD5, SHA1 or SHA256)
do_notice_sha256: true
```

- `download_history` defines how many days of indicators to initially download. Once the initial download is complete, the integration will run at the next interval and only pull changes back to the last successful download. If a download fails, or the `download_history` setting is changed, the next download will pull all indicators as defined by the `download_history`.
- `max_history` defines how many days of indicators to store in the local database.
- `use_history` defines how many days of indicators to use in the intel file.
- `min_confidence_score_use` defines the minimum confidence score an indicator must have to be included in the intel file.
- `min_confidence_score_download` defines the minimum confidence score an indicator must have to be downloaded from Mandiant. Note that Mandiant frequently updates its confidence scores for indicators, so configure this setting well below the `min_confidence_score_use`. If an indicator's confidence score is changed and downgraded below this setting, the latest indicator will not be pulled from Madiant, and the indicator record in the local database will retain the last downloaded confidence score until the `max_history` is met and it's aged out.
- `exclude_os_indicators` allows the download of open source indicators. This setting only applies to downloading new indicators. Once the indicator is downloaded, it will remain in the local database and in use until it no longer meets the `use_history` setting. It will remain in the local database until the `max_history` is met and it's aged out.

### 5.6.2.2 Intel log

This integration will enrich the intel.log with content like the following:

```
{
  "@path":"intel",
  "@sensor":"Lab-AP200",
  "@timestamp":"2023-01-06T05:13:38.841292Z",
  "ts":"2023-01-06T05:13:38.841292Z",
  "uid":"CNh51N3dSRfMZG1Pt4",
  "id.orig_h":"195.133.40.86",
  "id.orig_p":64910,
  "id.resp_h":"192.168.13.20",
  "id.resp_p":80,
  "seen.indicator":"77.247.181.165",
  "seen.indicator_type":"Intel::ADDR",
  "seen.where":"Conn::IN_ORIG",
  "matched": [
    "Intel::ADDR"
  ],
  "sources": [
    "blocklist_de",
    "cinsscore_ci_badguys",
    "blocklist_net_ua",
    "Mandiant",
    "dshield_block"
  ],
}
```

If the ExtendIntel Zeek package is loaded, the intel.log will be enriched with additional content like the following: (all indicators will not have all fields)

```
{
  "confidence": [99],
  "desc": [
    "Mandiant Threat Intellegence"
    ]
  "lastseen": [
    "2023-01-03T16:10:54Z"
    ],
  "firstseen": [
    "2021-03-20T10:10:01Z"
    ],
  "url": [
    "https://advantage.mandiant.com/"
    ],
  "reports": [
    "ID:23-00000242, Type:News Analysis"
  ],
  "campaigns": [],
  "associated": [
    "ID:threat-actor--b7e371c2-724e-5ffa-9e3c-9b1410513c27, Name:FIN13; ID:threat-
↪actor--8211bc17-9216-5e83-b54d-d1b04add12f3, Name:APT28; ID:threat-actor--7a39953e-
↪0dae-569a-9d49-d52a4a8865b1, Name:APT29; ID:threat-actor--2f0ab36a-02a6-59f7-ac23-
↪bcd824cc7c8e, Name:FIN4"
  ],
  "category": [
    "exploit",
    "exploit/vuln-scanning, exploit"
  ],
}
```

### 5.6.3 FireEye iSIGHT Threat Intelligence

Configure the FireEye iSIGHT Threat Intelligence integration to set how frequently the integration runs, how much history to initially download, how much history to use in an Intel file, and how much history to maintain in the SQLite DB. This integration uses the Mandiant Threat Intelligence v2 API.

**do_notice**

The `do_notice` flag can be set based on the indicator type. It is set in the DB based on the settings when the indicator is downloaded, and is updated in the intel file each time it is written.

**Tip:** By default, only MD5 hash support is enabled on a Corelight Sensor. It is recommended that you use only one hash type. If you plan on using another hash type, update the configuration and enable the appropriate package on the sensor.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

#### 5.6.3.1 Settings

```
fireeye:
  enabled:                          false
  interval_hours:                   1
  public_key:
  private_key:
  download_history:                 90  # days to download initially (max 90)
  max_history:                      365 # days to keep in the database
  use_history:                      180 # days to write to the intel file
  accept_version:                   "2.6"
  debug:                            false
  # Enable indicators below
  indicator_type_sender_address:    true
  do_notice_sender_address:         true
  indicator_type_source_domain:     true
  do_notice_source_domain:          true
  indicator_type_source_ip:         true
  do_notice_source_ip:              true
  indicator_type_filename:          true
  do_notice_filename:               true
  indicator_type_md5:               true
  do_notice_md5:                    true
  indicator_type_sha1:              false
  do_notice_sha1:                   true
  indicator_type_sha256:            false
  do_notice_sha256:                 true
  indicator_type_fuzzy_hash:        false
  do_notice_fuzzy_hash:             true
  indicator_type_user_agent:        true
  do_notice_user_agent:             true
  indicator_type_cidr:              true
  do_notice_cidr:                   true
  indicator_type_domain:            true
  do_notice_domain:                 true
  indicator_type_ip:                true
  do_notice_ip:                     true
  indicator_type_url:               true
  do_notice_url:                    true
```

#### 5.6.3.2 Intel log

This integration will enrich the intel.log with content like the following:

```
  {
    "@path":"intel",
    "@sensor":"Lab-AP200",
    "@timestamp":"2023-01-06T05:13:38.841292Z",
    "ts":"2023-01-06T05:13:38.841292Z",
    "uid":"CNh51N3dSRfMZG1Pt4",
    "id.orig_h":"195.133.40.86",
    "id.orig_p":64910,
    "id.resp_h":"192.168.13.20",
    "id.resp_p":80,
    "seen.indicator":"77.247.181.165",
    "seen.indicator_type":"Intel::ADDR",
```

(continues on next page)

(continued from previous page)

```
    "seen.where":"Conn::IN_ORIG",
    "matched": [
      "Intel::ADDR"
    ],
    "sources": [
      "blocklist_de",
      "cinsscore_ci_badguys",
      "blocklist_net_ua",
      "Mandiant",
      "dshield_block"
    ],
  }
```

### 5.6.4 icannTLD Zeek script

icannTLD is a Zeek script that uses the official ICANN Top-Level Domain (TLD) list to extract the relevant information from a DNS query and enrich the DNS log with that information. It can also mark whether it's trusted or not. The source of the ICANN TLDs can be found here: https://publicsuffix.org/list/effective_tld_names.dat.

Today, anyone can create a TLD and ICANN updates the list several times a day, as changes are made.

TLDs are generally split into two categories:

- ccTLDs are Country Code TLDs, such as .us, .jp and .uk
- gTLDs are Generic TLDs and include the traditional names .com, .net, and .org. Generic TLDs also include the new TLDs such as .info, .city, .microsoft, etc.

**As of December 2022, there are 6887 Top-Level Domains that can include up to 4 parts.**

- 19.2% (1,322) TLDs only contain one part (i.e. .com)
- 52.2% (3,597) TLDs contain two parts (i.e. mo.us)
- 28.5% (1,964) TLDs contain three parts (i.e. k12.mo.us)
- 0.1% (4) TLDs contain four parts (i.e. pvt.k12.ma.us)

As a result, any method of identifying TLDs without using the ICANN TLD database, i.e. regex, will miss identify over 80% of them.

**Tip:** The Trusted Domains list is a custom list, created by the user, to filter domains during searches.

#### 5.6.4.1 Script functions

icannTLD parses every DNS query and adds the following fields to the DNS Log.

Table 1: New DNS Log Fields

| Field | Value | Description |
|---|---|---|
| icann_tld | | This is the Top-Level Domain based on the official list of TLDs from ICANN. |
| icann_domain | | This is the Domain based on the official list of TLDs from ICANN. |
| icann_host_subdomain | | This is the remaining nodes of the query after the domain has been removed. In some cases this is the subdomain, in other cases it's the host name, and in others it's host name and subdomain. |
| is_trusted_domain | true/false | Each query is marked true or false based on the icann_domain and a custom *trusted_domains.dat* file. |

Corelight-update can generate the required Input files needed for the icannTLD Zeek script. However, the optional trusted domain list is not generated. See https://github.com/corelight/icannTLD for more details.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

The icannTLD settings:

```
icann_tld:
  enabled:         true
  interval_hours:  0
  url:             "https://publicsuffix.org/list/effective_tld_names.dat"
```

### 5.6.5 Maxmind GeoIP

Corelight physical and virtual sensors include a GeoIP database and are not updated with Corelight-update. This section only applies to Software Sensors.

You can sign up for free and get a license key from https://www.maxmind.com/en/geolite2/signup. Once you have an AccountID and LicenseKey, enter them in the `geoip` configuration below. You can also edit the GeoIP advanced configuration if you want to change additional settings. The GeoIP advanced configuration is in the Global Configuration and Policy settings file located here: `/etc/corelight-update/global/config.yaml`

#### 5.6.5.1 GeoIP settings

**Tip:** If you are running Corelight-update on the same host as a Corelight Software Sensor, the default location the sensor looks for the GeoIP database is `/usr/share/GeoIP/`

The GeoIP settings:

```
geoip:
  account_id:          0
  license_key:         ""
  database_directory:  "/var/corelight-update/files/all/geoip"
```

#### 5.6.5.2 Maxmind configuration settings

If you need to change more settings than listed above, you can edit the Maxmind configuration file as needed.

**Tip:** The Maxmind configuration file is located here: `/etc/corelight-update/global/GeoIP.conf`

### 5.6.6 Tenable.sc

The configuration required for Tenable Security Center is minimal.

- Each severity and pluginType must be listed.
- Provide the host address and port of the local TenableSC instance.

There is no need to set the integration interval more frequently than the frequency Tenable.SC is scanning the network.

If the `interval_hours` is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

Once downloaded, this data will be used to create an Input Framework file that can be used by a Zeek script to generate new logs, or enrich existing logs, such as the **suricata_corelight.log**.

The input file will be published with any other input files from other integrations (if there are any). If "input" is enabled in the "push_content" settings, the file will automatically get pushed to the Fleet Manager policy and/or all sensors in the policy. See *Push content settings* for more details.

**Attention:** The Nessus (Tenable Security Center) user you're using to provide an `access_key` and `secret_key` must have Security Management rights. Do not use an admin user.

#### 5.6.6.1 Settings

```
tenable_sc:
  enabled:                          false
  interval_hours:                   24
  access_key:
  secret_key:
  severity:                         "4,3,2,1"
  pluginType:                       "Active,Passive,Event"
  address:
  port:                             443
  request_limit:                    50000
```

#### 5.6.6.2 Input file

The following is a sample input file created by this integration, using tab-separated values.

```
#fields ip      hostname   host_uid    machine_domain    os_version    source       ␣
↪cve_list
192.168.2.186   mbp        -           -                 -             Tenable.SC   ␣
↪CVE-2021-1234,CVE-2021-4321
192.168.2.133   mbp        -           -                 -             Tenable.SC   ␣
↪CVE-2021-1234,CVE-2021-4321
```

(continues on next page)

(continued from previous page)

```
3.19.25.148     f5        -           -                 -             Tenable.SC   ␣
↪CVE-2020-5902
192.168.12.212  skynet    -           -                 -             Tenable.SC   ␣
↪CVE-2020-12313,CVE-2016-1585,CVE-2022-36227,CVE-2022-45884,CVE-2023-22995,CVE-2022-
↪38457
```

**Important:** To use this file, an additional Zeek package is required on the sensor.

#### 5.6.6.3 suricata_corelight log

A typical suricata_corelight.log provides content like the example below:

```
  {
    _path: suricata_corelight
    _system_name: Lab-AP200
    _write_ts: 2023-08-01T01:22:20.096550Z
    alert.action: allowed
    alert.category: Attempted Information Leak
    alert.gid: 1
    alert.metadata: [ [-]
      created_at:2023_07_28
      updated_at:2023_07_28
    ]
    alert.rev: 1
    alert.severity: 2
    alert.signature: **CONTROL** curl User-Agent Outbound CVE-2020-12313
    alert.signature_id: 1000000
    community_id: 1:hfVPB4FWl48hOvuIzwyVBvzWBwY=
    flow_id: 11005288195832
    id.orig_h: 192.168.12.212
    id.orig_p: 48086
    id.resp_h: 3.160.22.77
    id.resp_p: 80
    id.vlan: 12
    pcap_cnt: 0
    service: http
    suri_id: Sp5Hxvr0blDf
    ts: 2023-08-01T01:22:20.092276Z
    tx_id: 0
    uid: CUWMCe4TJo8pS41Rnj
  }
```

If the Zeek package is loaded, the suricata_corelight.log will be enriched with additional content provided by the integration, like the example below:

```
  {
    orig_vulnerable_host.cve: CVE-2020-12313
    orig_vulnerable_host.host_uid:
    orig_vulnerable_host.hostname: skynet
    orig_vulnerable_host.os_version:
    orig_vulnerable_host.source: Tenable.SC
  }
```

**Note:** Field names begin with "orig" or "resp" to identify which host is referenced.

### 5.6.7 AlienVault Open Threat Exchange

The main settings for the AlienVault OTX integration determines how frequently the integration runs, how much history to initially download, how much history to use in an Intel file, and how much history to keep in the SQLite DB.

The initial download will retrieve OTX threat intel "pulses" back to the configured days set in the 'download_history' setting. Each consecutive download will only contain new pulses since the last successful download. If you change the 'download_history' setting, the integration resets, and on the next run it will retrieve all pulses back to the new setting.

**do_notice**

The `do_notice` flag can be set based on the indicator type. It is set in the DB based on the settings when the indicator is downloaded and is updated in the intel file each time it is written.

**Tip:** By default, only MD5 hash support is enabled on a Corelight Sensor. It is recommended that you use only one hash type. If you plan on using another hash type, update the configuration and enable the appropriate package on the sensor.

If the 'interval_hours' is set to 0, the integration will attempt to download additional content each time the Corelight-update service runs. See *Configuration settings*

#### 5.6.7.1 Settings

```
alienvault_otx:
  enabled:                          false
  interval_hours:                   0
  url:                              "https://otx.alienvault.com"
  api_key:
  debug:                            false
  request_limit:                    10000
  download_history:                 90  # days to download initially (max 90)
  max_history:                      365 # days to keep in the database
  use_history:                      90 # days to write to the intel file
  # Enable indicator types below
  indicator_type_url:               true
  do_notice_url:                    false
  indicator_type_hostname:          true
  do_notice_hostname:               false
  indicator_type_domain:            true
  do_notice_domain:                 false
  indicator_type_ipv4:              true
  do_notice_ipv4:                   false
  indicator_type_md5:               true
  do_notice_md5:                    false
  indicator_type_sha1:              false
  do_notice_sha1:                   false
  indicator_type_sha256:            false
```

(continues on next page)

(continued from previous page)

```
do_notice_sha256:                 false
indicator_type_imphash:           false
do_notice_imphash:                false
indicator_type_sslcert:           true
do_notice_sslcert:                false
indicator_type_email:             false
do_notice_email:                  false
```

#### 5.6.7.2 Intel log

This integration will enrich the intel.log with content like the following:

```
{
  "@path":"intel",
  "@sensor":"Lab-AP200",
  "@timestamp":"2023-01-06T05:13:38.841292Z",
  "ts":"2023-01-06T05:13:38.841292Z",
  "uid":"CNh51N3dSRfMZG1Pt4",
  "id.orig_h":"195.133.40.86",
  "id.orig_p":64910,
  "id.resp_h":"192.168.13.20",
  "id.resp_p":80,
  "seen.indicator":"77.247.181.165",
  "seen.indicator_type":"Intel::ADDR",
  "seen.where":"Conn::IN_ORIG",
  "matched": [
    "Intel::ADDR"
  ],
  "sources": [
    "blocklist_de",
    "cinsscore_ci_badguys",
    "blocklist_net_ua",
    "Mandiant",
    "dshield_block"
  ],
}
```

### 5.6.8 Zeek package management

Corelight-update implements some basic package management functions, similar to the Zeek Package Manager (ZKG). https://docs.zeek.org/projects/package-manager/en/stable/

Corelight-update Zeek Package Management can:

- Build package bundles from a manifest file by downloading packages from the Internet.
- Build package bundles from a manifest file in offline mode.
- Push package bundles, built by Corelight-update, to Fleet Manager policies and/or sensors.
- Push package bundles, built off-box, to Fleet Manager policies and/or sensors.
- Push Corelight-signed package bundles to all sensors except Software Sensor v1.x sensors.

Corelight-update only generates package bundles from a manifest file. While Corelight-update can push package bundles that are created by other sources, it does not install packages locally or edit existing bundles.

**Warning:** Enabling "offline_mode" only prevents downloading the Zeek Package Index. If a URL is provided to a package repo in the manifest file, it still attempts to clone it.

The policy settings for Zeek Package Management are:

```
# Push Content to Sensors
push_content:
  package_bundle: false

# Enable Corelight Package Management
# Creates a package bundle for |cs|s
# Must be disabled to push external bundles
package_management:
  enabled:                     false
  offline_mode:                false
  manifest_file:               "bundle.manifest"
  bundle_name:                 "corelight.bundle" # Located in global-bundle or local-
↪bundle
  signed:                      false              # for externally created bundles only
```

The inventory settings for pushing Zeek Packages are:

```
# push package bundle to this sensor
bundle:          true
bundle_path:     "/etc/corelight/corelight.bundle"     # software sensors ONLY
```

**ZKG and Software Sensor v1.x**

Pushing a package bundle to a Software Sensor v1.x uses SCP and requires a path to place the bundle. After Corelight-update pushes a package bundle, it uses ZKG on the sensor to install the packages.

For details on how to install and setup ZKG on a Software Sensor v1.x, see *Zeek Package Manager (ZKG)*

### 5.6.8.1 Create and push a package bundle

To create and push a package bundle:

1. **Enable** `package_management` in the policy configuration.
2. Set the name of the manifest file. For example, `manifest_file:  bundle.manifest`
3. Place a manifest file in the policy configuration folder.
4. Set `push_package_bundle:  true` in the policy.
5. Ensure `bundle:  true` in the inventory file for the desired sensors.
6. If the manifest file changes, a new bundle will automatically be created and pushed each time the Corelight-update service runs.
   - Optionally, force create and push a bundle with the CLI command `corelight-update -b <policy name>`

#### 5.6.8.2 Push external package bundles

To push a package bundle created outside of Corelight-update:

1. **Disable** `package_management` in the policy configuration
2. Set the name of the bundle. For example, `bundle_name:  corelight.bundle`
3. Place the package bundle in the `global-bundle` or `local-bundle` folder
   - A package bundle in `local-bundle` takes precedence
4. Set `push_content:  package_bundle:  true` in the policy
5. Ensure `bundle:  true` in the inventory file for the desired sensor
6. External bundles are not automatically pushed to sensors. They must be pushed with the CLI command `corelight-update -b <policy name>`

**Attention:** Some integrations, such as Tenable.sc, Mandiant Threat Inteligence, and icannTLD require an additional Zeek script to be loaded on the sensors. See *Zeek package management*. If you enable the integration, Corelight-update will upload the input file to the sensor. But if the required script isn't available on the sensor, the input data won't be used.

CHAPTER

# SIX

# REFERENCES

## 6.1 CLI commands

**Warning:** When updating from a full or partial configuration, any config section provided must have all none-zero fields provided. Any missing fields will be updated to their zero value.

### 6.1.1 CLI help output

To view the available CLI Commands, use `corelight-update -h`

```
Options:
    -b  - Build and push a package bundle for select policies
    -d  - Turn on debug level logging
    -D  - Turn on debug level2 logging
    -f  - Force deploy existing content to a policy
    -h  - Print this help message
    -o  - Run once for a one or more policies
    -v  - Print Corelight-update version

**Run Corelight-update continuously for all defined policies**
  Usage: corelight-update

**Run Corelight-update once for all defined policies**
  Usage: corelight-update -o

**Force deploy existing content to select policies**
  Usage: corelight-update -f [policy1 policy2 ... policyN]

**Build and push a package bundle for select policies**
  Usage: corelight-update -b [policy1 policy2 ... policyN]

**Add Options**
  Usage: corelight-update add -policy [policy1 policy2 ... policyN]
  Usage: corelight-update add -policies [policy1 policy2 ... policyN]
    --policy    Add one or more policies
    --policies  Add one or more policies

**Import Options**
  Usage: corelight-update import -policy <policy name> -path <path to policy config> -
↪v0.23 -f
    --policy    Name of policy to import
```

(continues on next page)

(continued from previous page)

```
    --file      Path to config file to import (json or yaml)
    --path      Path to config file to import (json or yaml)
    --v0.23     Import legacy config file from v0.23
    -f          Force import - overwrite existing policy

**Remove Options**
  Usage: corelight-update remove -policy [policy1 policy2 ... policyN]
  Usage: corelight-update remove -policies [policy1 policy2 ... policyN]
  Usage: corelight-update remove -all-policies
    --policy        Name or names of policies to remove
    --policies      Name or names of policies to remove
    --all-policies  Remove all policies

**Show Options**
  Usage: corelight-update show -policies
  Usage: corelight-update show -policy <policy name> [-json|-yaml] [-file]
  Usage: corelight-update show -global [-json|-yaml] [-file]
  Usage: corelight-update show -network [-json|-yaml]
    --policies   List all policies
    --policy     Name of Policy to print
    --global     Print Global Config
    --network    Print Network Config
    --json|JSON  Print in JSON format
    --yaml|YAML  Print in YAML format - (Default)
    --file       File path to save output to
    --path       File path to save output to

**Update Options**
  Usage: corelight-update update -global -path <path to global config>
  Usage: corelight-update update -global-setting [setting1=value1 setting2=value2 ...␣
↪settingN=valueN]
  Usage: corelight-update update -global-settings [setting1=value1 setting2=value2 ...
↪ settingN=valueN]
  Usage: corelight-update update -network-setting [setting1=value1 setting2=value2 ...
↪ settingN=valueN]
  Usage: corelight-update update -network-settings [setting1=value1 setting2=value2 ..
↪. settingN=valueN]
  Usage: corelight-update update -policy <policy name> -path <path to policy config>
    --global            Update Global Config from a yaml or json config file
    --global-setting    Update Global Config setting from a key value pair
    --global-settings   Update Global Config setting from a key value pair
    --policy            Update a Policy Config from a yaml or json config file
    --file              Path to config file to import (json or yaml)
    --path               Path to config file to import (json or yaml)

**Encrypt Options**
  To encrypt passwords before they are stored in the 'encrypted_pass' field of a␣
↪policy
  Usage: corelight-update encrypt <string to encrypt>
```

## 6.2 Corelight-update service

When Corelight-update is installed, in addition to a `corelight-update.service`, a system user and group are automatically created. The service runs as the system user `corelight-update`. However, it's disabled by default. To run Corelight-update as a service, enable the service and start it.

1. Enable the service:

   ```
   sudo systemctl enable corelight-update.service
   ```

2. Start the service:

   ```
   sudo systemctl start corelight-update.service
   ```

3. To view the status of the service:

   ```
   systemctl status corelight-update.service
   ```

4. To monitor the logs from the service: The `-f` option makes the command follow the log until it's canceled.

   ```
   sudo journalctl -f -u corelight-update
   ```

## 6.3 Using a proxy with Corelight-update

If the Corelight-update host requires use of a network proxy to access and download content, you can configure the Corelight-update host service or user session to provide the proxy location using the `HTTPS_PROXY` or `https_proxy` environment variables.

### 6.3.1 Update the service definition

When running Corelight-update as a service, it will automatically use the `HTTPS_PROXY` or `https_proxy` environment variables when set. Corelight-update will not use any HTTP proxy variables. The proxy location can be added to the service definition.

Update the service definition using override.conf.

1. Use `systemctl` to create an override.conf.

   ```
    $ sudo systemctl edit corelight-update.service
   ```

2. Create a `[Service]` section in the `override.conf`, and set the `HTTPS_PROXY` environment variable. For example:

   ```
   [Service]
   Environment="HTTPS_PROXY=https://proxy.example.com:443"
   ```

3. Save the changes. You can review the `override.conf` in the path `/etc/systemd/system/corelight-update.service.d`

4. Reload systemd.

   ```
   $ sudo systemctl daemon-reload
   ```

5. Restart the Corelight-update service.

```
$ sudo systemctl restart corelight-update
```

### 6.3.2 Update the user environment

The options to set a proxy can vary based on the OS distribution being used. Corelight-update will automatically use the `HTTPS_PROXY` or `https_proxy` environment variables when set. Corelight-update will not use any HTTP proxy variables.

In general, you can set the proxy environment variables at the host and user level.

- In Red Hat, update `/etc/profile` to set the proxy at the host level for users.
- In Ubuntu, update `/etc/environment` to set the proxy at the host level for users.
- For user accounts, update the user's shell profile to set the proxy.

For example:

```
$ sudo vi /etc/profile

export https_proxy="https://proxy.example.com:443"
export HTTPS_PROXY="https://proxy.example.com:443"
```

## 6.4 Order of operations

The order of operations for every interval starts with:

1. Read the global policy configuration and each individual policy configuration.
2. Process the global tasks.
3. Process each policy, and push content for that policy.

### 6.4.1 Process global tasks

See *Configuration settings* for configuration options.

1. Process enabled integrations.
2. Download remote Suricata config files and store the in `/etc/corelight-update/global/`.
3. Download new content and update the Global Source Cache.
4. Remove content from the global cache for sources that are no longer configured.

### 6.4.2 Process policy tasks

See *Policy configuration* for configuration options

1. Copy local suricata rulesets from `/etc/corelight-update/configs/<policy>/local-suricata/` to the working directory.
2. Copy global suricata rulesets from `/etc/corelight-update/global/global-suricata/` to the working directory.

3. Copy local intel files from `/etc/corelight-update/configs/<policy>/local-intel/` to the working directory.
4. Copy global intel files from `/etc/corelight-update/global/global-intel/` to the working directory.
5. Remove content from the policy cache for sources that are no longer configured.
6. Download new content from policy sources.
7. Add default Input files to `/etc/corelight-update/configs/<policy>/local-input/` (if enabled - only runs once)
8. Process enabled integrations based on their intervals. See *Third-party integrations settings*
9. Process Input files and update the statefile.
10. Process Suricata rulesets.
    1. Collect ruleset files
        1. Collect new source content and copy it to the suricata working directory.
            - Check the global cache first.
            - If not in the global cache, download new content directly and update the policy level cache.
        2. Check for global `.rules` or `.rules.tar.gz` files in `/etc/corelight-update/global/global-suricata/` and extract/copy them to the suricata working directory.
        3. Check for local `.rules` or `.rules.tar.gz` files in `/etc/corelight-update/configs/<policy>/local-suricata/` and extract/copy them to the suricata working directory.
    2. Merge all of the rulesets into a single ruleset, ignoring any ruleset file identified with *File filters* in the following:
        - Corelight recommended disable.conf (if enabled)
        - global disable.conf (if it exists)
        - policy disable.conf (if it exists)
    3. If enabled, process Corelight recommended **disable.conf**, **enable.conf** and **modify.conf** files in that order.
    4. If they exist, process global **disable.conf**, **enable.conf** and **modify.conf** files in that order.
    5. If they exist, process policy **disable.conf**, **enable.conf** and **modify.conf** files in that order.
    6. If enabled, extract selected atomic rules from the Suricata ruleset and generate a Zeek Intel file.
    7. If enabled and Suricata is installed on the same host, test the new ruleset with Suricata in test mode (see *Suricata configuration* for details).
    8. Publish the new Suricata ruleset - **suricata.rules**.
11. Process Intel files
    1. Check for global intel files in `/etc/corelight-update/global/global-intel/`, and copy them to the intel working directory.
    2. Check for local intel files in `/etc/corelight-update/configs/<policy>/local-intel`, and copy them to the intel working directory.
    3. Evaluate all of the global, local, and integration intel files from all sources. If `disable.intel` is available, evaluate and remove indicators. Merge and dedupe results into a single file.
    4. Publish the new intel file - **intel.dat**

### 6.4.3 Push content for policies

Corelight-update deploys content updates in a specific order:

1. Push new Intel files.
2. Push new Suricata ruleset.
3. Push new Zeek Package bundle.
4. Push new Input files

By default, Corelight-update will push updates to the sensors concurrently. Corelight-update will open a connection to multiple sensors in a policy, push updated content, and cycle to the next sensor, up to the global configuration setting `parallel_push_limit`. See *Configuration settings*.

**Tip:** Corelight-update only attempts to push new content to sensors. You can manually force a push of all existing content to a group of sensors using the *CLI commands*.

## 6.5 Build test process

Corelight-update is currently tested with Docker to ensure it successfully installs on the following operating systems:

- image: centos:7
- image: rockylinux:8
- image: rockylinux:9
- image: registry.access.redhat.com/ubi8/ubi
- image: debian:10
- image: debian:11
- image: ubuntu:18.04
- image: ubuntu:20.04
- image: ubuntu:22.04
- image: ubuntu:22.04 (arm64)
- image: amazonlinux:2

## 6.6 System requirements

The minimum system requirements are:

- An x86_64 or ARM64 processor.
- 4 GB memory.
- A host running a Linux OS.
- Network connectivity to the Internet, or to an internal-facing threat intelligence data repository.
- To push content to your sensors, or to Fleet Manager, network connectivity to the management interface is required.

## 6.7 Commonly used Suricata rulesets

Any source that can be downloaded in the standard Suricata ruleset format, and does not require authentication, can be added to the list of sources. Here is a list of common Suricata ruleset sources. Just verify the URL, modify as needed, and add it to your list of sources.

- **Corelight Labs Suricata Rules:** `https://feed.corelight.com/corelight.rules`
- **ET/Open:** `https://rules.emergingthreats.net/open/suricata-6.0/emerging.rules.tar.gz`
- **ET/Pro:** `https://rules.emergingthreatspro.com/<insert-et-pro-key-here>/suricata-6.0/etpro.rules.tar.gz`
- **oisf/trafficid:** `https://openinfosecfoundation.org/rules/trafficid/trafficid.rules`
- **ptresearch/attackdetection:** `https://raw.githubusercontent.com/ptresearch/AttackDetection/master/pt.rules.tar.gz`
- **scwx/enhanced:** `https://ws.secureworks.com/ti/ruleset/<insert-secret-code-here>/Suricata_suricata-enhanced_latest.tgz`
- **scwx/malware:** `https://ws.secureworks.com/ti/ruleset/<insert-secret-code-here>/Suricata_suricata-malware_latest.tgz`
- **scwx/security:** `https://ws.secureworks.com/ti/ruleset/<insert-secret-code-here>/Suricata_suricata-security_latest.tgz`
- **sslbl/ssl-fp-blacklist:** `https://sslbl.abuse.ch/blacklist/sslblacklist.rules`
- **sslbl/js3-fingerprints:** `https://sslbl.abuse.ch/blacklist/ja3_fingerprints.rules`
- **etnetera/aggressive:** `https://security.etnetera.cz/feeds/etn_aggressive.rules`
- **tgreen/hunting:** `https://raw.githubusercontent.com/travisbgreen/hunting-rules/master/hunting.rules`
- **malsilo:** `https://malsilo.gitlab.io/feeds/dumps/malsilo.rules.tar.gz`

## 6.8 Third-party configuration guides

### 6.8.1 MISP - Zeek export

An export of all attributes of a specific bro type to a formatted plain text file. By default only published and IDS flagged attributes are exported.

You can configure your tools to automatically download a file one of the Bro types.

```
https://misp/attributes/bro/download/all
https://misp/attributes/bro/download/ip
https://misp/attributes/bro/download/url
https://misp/attributes/bro/download/domain
https://misp/attributes/bro/download/ja3-fingerprint-md5
https://misp/attributes/bro/download/email
https://misp/attributes/bro/download/filename
https://misp/attributes/bro/download/filehash
https://misp/attributes/bro/download/certhash
https://misp/attributes/bro/download/software
```

To restrict the results by tags, use the usual syntax. Please be aware the colons (:) cannot be used in the tag search. Use semicolons instead (the search will automatically search for colons instead). To get ip values from events tagged tag1 but not tag2 use:

```
https://misp/attributes/bro/download/ip/tag1&&!tag2
```

Alternatively, it is also possible to pass the filters via the parameters in the URL. The format is as described below:

```
https://misp/attributes/bro/download/[type]/[tags]/[event_id]/[from]/[to]/
↪[last]
```

```
type: The Zeek type, any valid Bro type is accepted.  See below for a␣
↪mapping between Zeek and MISP types.
tags: To include a tag in the results just write its names into this␣
↪parameter. To exclude a tag prepend it with a '!'. You can also chain␣
↪several tag commands together with the '&&' operator. Please be aware the␣
↪colons (:) cannot be used in the tag search. Use semicolons instead (the␣
↪search will automatically search for colons instead).
event_id: Restrict the results to the given event IDs.
allowNonIDS: Allow attributes to be exported that are not marked as "to_ids".
from: 'Events with the date set to a date after the one specified in the␣
↪from field (format: 2015-02-15). This filter will use the date of the␣
↪event.'
to: 'Events with the date set to a date before the one specified in the to␣
↪field (format: 2015-02-15). This filter will use the date of the event.'
last: Events published within the last x amount of time, where x can be␣
↪defined in days, hours, minutes (for example 5d or 12h or 30m). This␣
↪filter will use the published timestamp of the event.
enforceWarninglist: All attributes that have a hit on a warninglist will be␣
↪excluded.
```

| Zeek Type | MISP Type |
|---|---|
| **all:** | All types listed below. |
| **ip:** | *ip-src, ip-dst, ip-src\|port, ip-dst\|port, domain\|ip* |
| **url:** | url |
| **domain:** | hostname, domain, domain\|ip |
| **ja3-fingerprint-md5:** | ja3-fingerprint-md5 |
| **email:** | email, email-src, email-dst, target-email |
| **filename:** | filename, email-attachment, attachment, filename\|md5, filename\|sha1, filename\|sha256, malware-sample, pdb |
| **filehash:** | md5, sha1, sha256, authentihash, ssdeep, imphash, pehash, impfuzzy, sha224, sha384, sha512, sha512/224, sha512/256, tlsh, filename\|md5, filename\|sha1, filename\|sha256, filename\|authentihash, filename\|ssdeep, filename\|imphash, filename\|pehash, filename\|impfuzzy, filename\|sha224, filename\|sha384, filename\|sha512, filename\|sha512/224, filename\|sha512/256, filename\|tlsh, malware-sample |
| **certhash:** | x509-fingerprint-sha1 |
| **software:** | user-agent |

The keywords false or null should be used for optional empty parameters in the URL.

For example, to retrieve all attributes for event #5, including non IDS marked attributes too, use the following line:

```
https://misp/attributes/text/download/all/null/5/true
```

### 6.8.2 ThreatQ - Zeek exports

These steps explain how to export Zeek indicators for use with an external threat detection system. Follow these instructions to export your data.

1. Select the **Settings icon > Exports**.

   The Exports page appears with a table listing all exports in alphabetical order.

2. Click **Add New Export**.

   The Connection Settings dialog box appears.

3. Enter an **Export Name**.

4. Click **Next Step**.

   The Output Format dialog box appears.

5. Provide the following information:

| FIELD | VALUE |
|---|---|
| Which type of information would you like to export? | Indicators |
| Output Type | text/plain |
| Special Parameters | indicator.status=Active&indicator.deleted=N |

**Note:** You can edit the output format. This includes the ability to remove unwanted indicator types.

6. Under Output Format Template, enter:

```
#fields{$tab}indicator{$tab}indicator_type{$tab}meta.source{$tab}meta.url
{foreach $data as $indicator}
{$indicator_type=""}
{$source_found=0}
{if $indicator.type eq "CIDR Block"}{$indicator_type="Intel::SUBNET"}{/if}
{if $indicator.type eq "IP Address"}{$indicator_type="Intel::ADDR"}{/if}
{if $indicator.type eq "URL"}{$indicator_type="Intel::URL"}{/if}
{if $indicator.type eq "Email Address"}{$indicator_type="Intel::EMAIL"}{/
→if}
{if $indicator.type eq "FQDN"}{$indicator_type="Intel::DOMAIN"}{/if}
{if $indicator.type eq "MD5"}{$indicator_type="Intel::FILE_HASH"}{/if}
{if $indicator.type eq "SHA-1"}{$indicator_type="Intel::FILE_HASH"}{/if}
{if $indicator.type eq "SHA-256"}{$indicator_type="Intel::FILE_HASH"}{/if}
{if $indicator.type eq "SHA-384"}{$indicator_type="Intel::FILE_HASH"}{/if}
{if $indicator.type eq "SHA-512"}{$indicator_type="Intel::FILE_HASH"}{/if}
{if $indicator.type eq "Filename"}{$indicator_type="Intel::FILE_HASH"}{/
→if}
{if $indicator_type ne ""}
{$indicator.value}{$tab}{$indicator_type}{$tab}{foreach $indicator.
→Sources item=source name=Sources}{if $smarty.foreach.Sources.first ==␣
→true}
{$source.value}{$source_found=1}{/if}{/foreach}{if $source_found == 0}-{/
→if}
```

(continues on next page)

(continued from previous page)

```
{$tab}https://{$http_host}/indicators/{$indicator.id}/details
{/if}
{/foreach}
```

7. Click **Save Settings**.
8. Under **On/Off**, toggle the switch to enable the export.

When finished, use the URL to download the intel data in Zeek format.

## 6.8.3 Zeek Package Manager (ZKG)

### 6.8.3.1 Quickstart guide

These instructions are intended for installations of ZKG on the same host as a Software Sensor.

### 6.8.3.2 Dependencies

- Python 3.6+
- git: https://git-scm.com
- GitPython: https://pypi.python.org/pypi/GitPython
- semantic_version: https://pypi.python.org/pypi/semantic_version
- btest: https://pypi.python.org/pypi/btest

Note that following the ZKG installation process via pip3 will automatically install its dependencies for you.

### 6.8.3.3 Installation

It is recommended to use the latest version of pip3:

```
sudo python3 -m pip install --upgrade --upgrade-strategy only-if-needed pip
```

To install the latest release of ZKG on PyPi:

```
sudo python3 -m pip install zkg
```

### 6.8.3.4 Basic setup

ZKG supports four broad approaches for managing Zeek packages. These details represent one of those approaches and are specific for a Corelight Software Sensor running as root.

1. Create the directory for the ZKG configurations.

   ```
   sudo mkdir /root/.zkg
   ```

2. Create/Edit the file `/root/.zkg/config` and add the following contents:

```
[sources]
zeek = https://github.com/zeek/packages

[paths]
state_dir = /root/.zkg
script_dir = /etc/corelight
zeek_dist =
```

3. Run the following command to refresh the Zeek index and create the `/etc/corelight/packages` directory.

   ```
   sudo zkg refresh
   ```

4. Edit `/etc/corelight/local.zeek` and add the following line:

   ```
   @load /etc/corelight/packages
   ```

#### 6.8.3.5 Usage

Corelight-update will use ZKG to manage package bundles on a Software Sensor.

Check the output of zkg –help for an explanation of all available functionality of the command-line tool.